POLITECNICO DI TORINO

Master's Degree Course in Biomedical Engineering

Master's Degree Thesis

Evaluation of pediatric brain tissue segmentation from MRI using a clustering method

Supervisors
Gabriella Balestra, PhD
Samanta Rosati, PhD

Laia Movilla Galve
266909

August 2019

Written in LaTeX on September 3, 2019

For grey matter, there is no black and white. If you think in black and white, then you do not use enough brain functions.

Petek Kabakci

Contents

List of Figures vii

List of Tables xiii

Abstract xiv

Resum xv

Resumen xvi

Acknowledgements xvii

Acronyms xviii

1 Introduction 1

2 Materials and methods 3
2.1 Population . . . 3
2.2 MRI Acquisition . . . 4
2.3 Clustering methods methods . . . 4
2.3.1 k-means algorithm . . . 5
2.3.2 Region-growing algorithm . . . 6
2.3.3 Hierarchical clustering . . . 7
2.3.4 Statistical parametric mapping (SPM) . . . 8
2.4 Evaluation methods . . . 9
2.4.1 Sørensen-Dice index . . . 9
2.4.2 Pixel classification . . . 9
2.4.3 Visual analysis . . . 10
2.5 Initial code . . . 11

3 Implementation 17
3.1 Initial code evaluation . . . 17
3.1.1 Results for different thresholds . . . 17
3.1.2 Results for different versions . . . 20
3.2 Code combination . . . 22
3.2.1 Dice index for each slice . . . 23
3.2.2 Tissue recognition by slice . . . 24

3.2.3 Tissue recognition by slice - percentage . . . 26
3.2.4 Tissue recognition by row . . . 28
3.3 Threshold modification . . . 30
3.4 Hierarchical clustering . . . 32
3.5 Patient selection . . . 34
3.6 Dendrogram analysis . . . 37
3.7 *clust_image* function improvement . . . 40
3.8 Segmentation order . . . 43
3.9 Final results . . . 46

4 Conclusions and future work 47

Appendices 49

A *main_perArticolo* function 51

B *im_reorientation* function 55

C *calcolo_maschera_CSF1* function 57

D *calcolo_maschera_GM* function 61

E *calcolo_maschera_WM* function 67

F *creazione_matr_clustering* function 71

G *region_growing1* function 73

H *region_growing* function 75

I *scarta_pixel* function 77

J *calcolo_medie* function 79

K *creazione_maschere* function 81

L *clust_image* function 83

List of Figures

2.1 Display of the thirteen original patients' MRI (FLAIR mask) 4
2.2 Steps of the k-means algorithm 5
2.3 Steps of the region-growing algorithm 7
2.4 Example of a dendrogram 8
2.5 NeonateB's standard masks 10
2.6 Original version: code files 12
2.7 Example of a T1 (A), FLAIR (B) AND *T1*FLAIR* (C) of a neonate 13
2.8 Example of a T1 (A), T2 (B) AND $T2^*$**T1* (C) of a neonate 14

3.1 Dice indices for the GM mask with different binarizing thresholds . . 18
3.1a GM mask - threshold 0 18
3.1b GM mask - threshold 0.1 18
3.1c GM mask - threshold 0.2 18
3.1d GM mask - threshold 0.3 18
3.1e GM mask - threshold 0.4 18
3.1f GM mask - threshold 0.5 18
3.1g GM mask - threshold 0.6 18
3.1h GM mask - threshold 0.7 18
3.1i GM mask - threshold 0.8 18
3.1j GM mask - threshold 0.9 18
3.2 Dice indices for the WM mask with different binarizing thresholds . . 19
3.2a WM mask - threshold 0 19
3.2b WM mask - threshold 0.1 19
3.2c WM mask - threshold 0.2 19
3.2d WM mask - threshold 0.3 19
3.2e WM mask - threshold 0.4 19
3.2f WM mask - threshold 0.5 19
3.2g WM mask - threshold 0.6 19
3.2h WM mask - threshold 0.7 19
3.2i WM mask - threshold 0.8 19
3.2j WM mask - threshold 0.9 19
3.3 Dice indices for initial code versions 20
3.3a Neonate7 - initial code 20
3.3b NeonateA - initial code 20
3.3c NeonateB - initial code 20
3.3d NeonateC - initial code 20

3.3 Dice indices for initial code versions 21
3.3e NeonateD - initial code 21
3.3f NeonateE - initial code 21
3.3g NeonateF - initial code 21
3.3h NeonateG - initial code 21
3.3i NeonateI - initial code 21
3.3j NeonateK - initial code 21
3.3k NeonateL - initial code 21
3.3l NeonateM - initial code 21
3.3m NeonateN - initial code 21
3.4 Dice indices for each slice in both GM and WM masks 23
3.4a GM mask - threshold 0 23
3.4b GM mask - threshold 0.1 23
3.4c GM mask - threshold 0 23
3.4d GM mask - threshold 0.1 23
3.5 Pixel classification by slice for v3 24
3.5a Neonate7 - pixels by slice 24
3.5b NeonateA - pixels by slice 24
3.5c NeonateB - pixels by slice 24
3.5d NeonateC - pixels by slice 24
3.5 Pixel classification by slice for v3 25
3.5e NeonateD - pixels by slice 25
3.5f NeonateE - pixels by slice 25
3.5g NeonateF - pixels by slice 25
3.5h NeonateG - pixels by slice 25
3.5i NeonateI - pixels by slice 25
3.5j NeonateK - pixels by slice 25
3.5k NeonateL - pixels by slice 25
3.5l NeonateM - pixels by slice 25
3.5m NeonateN - pixels by slice 25
3.6 Pixel classification by slice for v3 (percentage) 26
3.6a Neonate7 - pixels by slice(%) 26
3.6b NeonateA - pixels by slice(%) 26
3.6c NeonateB - pixels by slice(%) 26
3.6d NeonateC - pixels by slice(%) 26
3.6 Pixel classification by slice for v3 (percentage) 27
3.6e NeonateD - pixels by slice(%) 27
3.6f NeonateE - pixels by slice(%) 27
3.6g NeonateF - pixels by slice(%) 27
3.6h NeonateG - pixels by slice(%) 27
3.6i NeonateI - pixels by slice(%) 27
3.6j NeonateK - pixels by slice(%) 27
3.6k NeonateL - pixels by slice(%) 27
3.6l NeonateM - pixels by slice(%) 27
3.6m NeonateN - pixels by slice(%) 27

3.7 Pixel classification by row for v3 . . . 28
3.7a Neonate7 - pixels by row . . . 28
3.7b NeonateA - pixels by row . . . 28
3.7c NeonateB - pixels by row . . . 28
3.7d NeonateC - pixels by row . . . 28
3.7 Pixel classification by row for v3 . . . 29
3.7e NeonateD - pixels by row . . . 29
3.7f NeonateE - pixels by row . . . 29
3.7g NeonateF - pixels by row . . . 29
3.7h NeonateG - pixels by row . . . 29
3.7i NeonateI - pixels by row . . . 29
3.7j NeonateK - pixels by row . . . 29
3.7k NeonateL - pixels by row . . . 29
3.7l NeonateM - pixels by row . . . 29
3.7m NeonateN - pixels by row . . . 29
3.8 Threshold modification results . . . 30
3.8a Neonate7 - v3 modified . . . 30
3.8b NeonateA - v3 modified . . . 30
3.8c NeonateB - v3 modified . . . 30
3.8d NeonateC - v3 modified . . . 30
3.8 Threshold modification results . . . 31
3.8e NeonateD - v3 modified . . . 31
3.8f NeonateE - v3 modified . . . 31
3.8g NeonateF - v3 modified . . . 31
3.8h NeonateG - v3 modified . . . 31
3.8i NeonateI - v3 modified . . . 31
3.8j NeonateK - v3 modified . . . 31
3.8k NeonateL - v3 modified . . . 31
3.8l NeonateM - v3 modified . . . 31
3.8m NeonateN - v3 modified . . . 31
3.9 Hierarchical clustering results . . . 33
3.9a Neonate7 - hierarchical . . . 33
3.9b NeonateA - hierarchical . . . 33
3.9c NeonateB - hierarchical . . . 33
3.9d NeonateC - hierarchical . . . 33
3.9e NeonateD - hierarchical . . . 33
3.9f NeonateE - hierarchical . . . 33
3.9g NeonateF - hierarchical . . . 33
3.9h NeonateG - hierarchical . . . 33
3.9i NeonateI - hierarchical . . . 33
3.9j NeonateK - hierarchical . . . 33
3.9 Hierarchical clustering results . . . 34
3.9k NeonateL - hierarchical . . . 34
3.9l NeonateM - hierarchical . . . 34
3.9m NeonateN - hierarchical . . . 34

3.10 Code version comparison . . . 34
3.11 Patients selection: GM and WM display . . . 35
3.11a Neonate7 - GM slices . . . 35
3.11b Neonate7 - WM slices . . . 35
3.11c NeonateA - GM slices . . . 35
3.11d NeonateA - WM slices . . . 35
3.11e NeonateB - GM slices . . . 35
3.11f NeonateB - WM slices . . . 35
3.11g NeonateC - GM slices . . . 35
3.11h NeonateC - WM slices . . . 35
3.11i NeonateD - GM slices . . . 35
3.11j NeonateD - WM slices . . . 35
3.11 Patients selection: GM and WM display . . . 36
3.11k NeonateE - GM slices . . . 36
3.11l NeonateE - WM slices . . . 36
3.11m NeonateF - GM slices . . . 36
3.11n NeonateF - WM slices . . . 36
3.11o NeonateG - GM slices . . . 36
3.11p NeonateG - WM slices . . . 36
3.11q NeonateI - GM slices . . . 36
3.11r NeonateI - WM slices . . . 36
3.11s NeonateK - GM slices . . . 36
3.11t NeonateK - WM slices . . . 36
3.11 Patients selection: GM and WM display . . . 37
3.11u NeonateL - GM slices . . . 37
3.11v NeonateL - WM slices . . . 37
3.11w NeonateM - GM slices . . . 37
3.11x NeonateM - WM slices . . . 37
3.11y NeonateN - GM slices . . . 37
3.11z NeonateN - WM slices . . . 37
3.12 Zone selection according to slice . . . 38
3.13 Dendrograms Neonate7 . . . 38
3.13a Dendrogram - zone 1.1 . . . 38
3.13b Dendrogram - zone 1.2 . . . 38
3.13c Dendrogram - zone 2.1 . . . 38
3.13d Dendrogram - zone 2.2 . . . 38
3.13e Dendrogram - zone 3.1 . . . 38
3.13f Dendrogram - zone 3.2 . . . 38
3.13 Dendrograms Neonate7 . . . 39
3.13g Dendrogram - zone 4.1 . . . 39
3.13h Dendrogram - zone 4.2 . . . 39
3.13i Dendrogram - zone 5.1 . . . 39
3.13j Dendrogram - zone 5.2 . . . 39
3.13k Dendrogram - zone 6.1 . . . 39
3.13l Dendrogram - zone 6.2 . . . 39

3.13m Dendrogram - zone 7.1 . . . 39
3.13n Dendrogram - zone 7.2 . . . 39
3.14 Pixel percentage - v3.0.1 . . . 40
3.14a Neonate7 - pixels percentage . . . 40
3.14b NeonateA - pixels percentage . . . 40
3.14c NeonateC - pixels percentage . . . 40
3.14d NeonateD - pixels percentage . . . 40
3.14e NeonateE - pixels percentage . . . 40
3.14f NeonateF - pixels percentage . . . 40
3.14 Pixel percentage - v3.0.1 . . . 41
3.14g NeonateG - pixels percentage . . . 41
3.15 Pixel percentage - v4.2.1 . . . 41
3.15a Neonate7 - pixels percentage . . . 41
3.15b NeonateA - pixels percentage . . . 41
3.15c NeonateC - pixels percentage . . . 41
3.15d NeonateD - pixels percentage . . . 41
3.15e NeonateE - pixels percentage . . . 41
3.15f NeonateF - pixels percentage . . . 41
3.15 Pixel percentage - v4.2.1 . . . 42
3.15g NeonateG - pixels percentage . . . 42
3.16 Pixel percentage - v4.4.1 . . . 42
3.16a Neonate7 - pixels percentage . . . 42
3.16b NeonateA - pixels percentage . . . 42
3.16c NeonateC - pixels percentage . . . 42
3.16d NeonateD - pixels percentage . . . 42
3.16e NeonateE - pixels percentage . . . 42
3.16f NeonateF - pixels percentage . . . 42
3.16 Pixel percentage - v4.4.1 . . . 43
3.16g NeonateG - pixels percentage . . . 43
3.17 Dice index comparison - function improvement . . . 43
3.17a Dice index - GM mask . . . 43
3.17b Dice index - WM mask . . . 43
3.18 Pixel percentage - v5 . . . 44
3.18a Neonate7 - pixels percentage . . . 44
3.18b NeonateA - pixels percentage . . . 44
3.18c NeonateC - pixels percentage . . . 44
3.18d NeonateD - pixels percentage . . . 44
3.18e NeonateE - pixels percentage . . . 44
3.18f NeonateF - pixels percentage . . . 44
3.18g NeonateG - pixels percentage . . . 44
3.19 Pixel percentage - v5.1 . . . 45
3.19a Neonate7 - pixels percentage . . . 45
3.19b NeonateA - pixels percentage . . . 45
3.19c NeonateC - pixels percentage . . . 45
3.19d NeonateD - pixels percentage . . . 45

3.19e NeonateE - pixels percentage 45
3.19f NeonateF - pixels percentage 45
3.19g NeonateG - pixels percentage 45

List of Tables

2.1 Pixel classification 10
2.2 Test subject selection 11

3.1 Code files for v3 22
3.2 Code files for v3 30
3.3 Hierarchical clustering versions 32
3.4 GM masks' dice index 46
3.5 WM masks' dice index 46

Abstract

Image segmentation in medical images such as magnetic resonances (MR) is extremely important for faster and more reliable diagnoses. Since nowadays most segmentation systems are manual, new automatic methods are being developed to achieve more consistent results.

Starting from a code which uses an atlas-free BTS (*afBTS*) algorithm to segment the brain MR images of neonates, this study aims to improve the obtained segmentation creating new code versions.

Four different clustering methods are going to be used: k-means algorithm, hierarchical clustering, region-growing algorithm and statistical parametric mapping (SPM). With these methods and other changes, twelve new versions are going to be developed. In order to assess the segmentations, the Sørensen-Dice index and the correct pixel recognition will be used.

Based on the results and conclusions obtained from this project, different alternatives and future developments will be proposed to deal with the weakest points in the new code versions.

Keywords: image segmentation, magnetic resonance, grouping, hierarchical clustering, dice index

Resum

La segmentació d'imatges mèdiques com les ressonàncies magnètiques (RM) és extremadament important per a diagnòstics més ràpids i fiables. Com avui dia la majoria dels sistemes de segmentació són manuals, s'estan desenvolupant nous mètodes automàtics per aconseguir resultats més consistents.

A partir d'un codi que utilitza un algorisme atlas-free BTS (*afBTS*) per segmentar les RM cerebrals de nounats, aquest estudi té com a objectiu millorar la segmentació obtinguda creant noves versions d'aquest codi.

S'utilitzaran quatre mètodes d'agrupació diferents: algorisme *k-means*, agrupament jeràrquic, algorisme de creixement regional i mapatge paramètric estadístic (SPM). Amb aquests mètodes i altres canvis, es crearan dotze noves versions. Per avaluar les segmentacions, s'utilitzaran l'índex Sørensen-Dice i el reconeixement i classificació de píxels.

Basant-se en els resultats i conclusions obtingudes d'aquest projecte, es proposaran diferents alternatives i desenvolupaments futurs per tractar els punts més febles en les noves versions de codi.

Keywords: segmentació d'imatges, ressonància magnètica, agrupació, agrupació jeràrquica, índex diu

Resumen

La segmentación de imágenes médicas como las resonancias magnéticas (RM) es extremadamente importante para diagnósticos más rápidos y fiables. Como hoy en día la mayoría de los sistemas de segmentación son manuales, se están desarrollando nuevos métodos automáticos para lograr resultados más consistentes.

A partir de un código que utiliza un algoritmo atlas-free BTS (*afBTS*) para segmentar las RM cerebrales de neonatos, este estudio tiene como objetivo mejorar la segmentación obtenida creando nuevas versiones de código.

Se utilizarán cuatro métodos de agrupación diferentes: algoritmo *k-means*, agrupamiento jerárquico, algoritmo de crecimiento regional y mapeo paramétrico estadístico (SPM). Con estos métodos y otros cambios, se crearán doce nuevas versiones. Para evaluar las segmentaciones, se utilizarán el índice Sørensen-Dice y el reconocimiento y clasificación de píxeles.

En base a los resultados y conclusiones obtenidas de este proyecto, se propondrán diferentes alternativas y desarrollos futuros para tratar los puntos más débiles en las nuevas versiones de código.

Keywords: segmentación de imágenes, resonancia magnética, agrupación, agrupación jerárquica, índice dice

Acknowledgements

I would like to thank professors Gabriella Balestra, PhD and Samanta Rosati, PhD for the guidance during the development of this thesis.

I would also like to thank my parents and brother for the unconditional love and support.

Acronyms

BSE	Brain Surface Extractor
BTS	Brain tissue segmentation
CSF	Cerebrospinal fluid
FN	False negative
FP	False positive
GM	Grey matter
ISODATA	Iterative self-organizing data analysis technique algorithm
MR	Magnetic resonance
MRI	Magnetic resonance imaging
NaN	Not a number
RG	Region Growing
ROI	Region of interest
SPM	Statistical parametric mapping
TN	True negative
TP	True positive
WM	White matter

Chapter 1

Introduction

The method of distinguishing a specific object from the surrounding background is defined as segmentation. When applied to the field of medical image analysis, it plays a crucial role in extracting targeted region of interest (ROI) and afterwards performing a monitorization on a disease progression.

Nowadays most segmentations are still done manually or with semi-automatic models; these methods are both time consuming and susceptible to inter-subject variability. Therefore, in the image segmentation research there is a major gap. New fully automatic segmentation models are being developed but all methods have their own advantages and disadvantages. The main improvement in automatic approaches is the elimination of the human intervention hence keeping a consistency in the obtained results. These methods, however, have not been able to deal with the great anatomical diversity present in human organs. Even with some flaws, the new segmentation models that are being designed are a major improvement to the old methods that were being used [1].

As far as brain magnetic resonance imaging (MRI) segmentation goes, the main goal is to divide the brain into cerebrospinal fluid (CSF), grey matter (GM) and white matter (WM). This is vital since many neurological diseases can be identified by their gradual alteration of the cellular environment which leads to macroscopic effects that can be seen in the magnetic resonances (MR). In the anatomical structures these changes can be in shape, size or image intensity [2]. In neonates, the brain segmentation is even harder to obtain because of the low spatial resolution, the severe partial volume effect, the high image noise, and the dynamic myelination and maturation processes [3].

This study is going to evaluate the feasibility of a segmentation method based on clustering applied to pediatric brain tissue. This will be done with an atlas-free BTS (*af*BTS) algorithm. The evaluation is going to be done mainly using the Dice index. Based on the obtained results new code versions will be done to hopefully improve the image segmentation.

Chapter 2

Materials and methods

In order to develop this thesis, different segmentation methods and evaluation methods were used to try to obtain a better brain segmentation for the different tests' subjects. In this chapter all the different methods and materials that were used throughout the developments of the thesis will be described for better comprehension of the project.

2.1 Population

For this study, a group of children which were younger than five years old were originally analysed. These test subjects had all undergone an MR examination with a 3 Tesla scanner between August 2018 and January 2019. All the original MRI were segmented using statistical parametric mapping (SPM) segmentation to obtain a standard segmentation.

After the first segmentation, two exclusion criteria were applied to the test subjects:

- The poor quality of the MR images
- The neurological disorders at clinical examination

Subsequently, thirteen neonates (younger than one month old) were initially considered for the study.

At some point during the development of the thesis the segmented masks were visually analysed to discern which of the Neonate could be used as a standard for evaluating the clustering code. This happened because some of the obtained results did not represent the reality. As a result, only seven of the thirteen initial test subjects were used in the last part of the study. To easily identify them

they were named as follows: Neonate7, NeonateA, NeonateC, NeonateD, NeonateE, NeonateF and NeonateG. The discarded patients were: NeonateB, NeonateI, NeonateK, NeonateL, NeonateM and NeonateN.

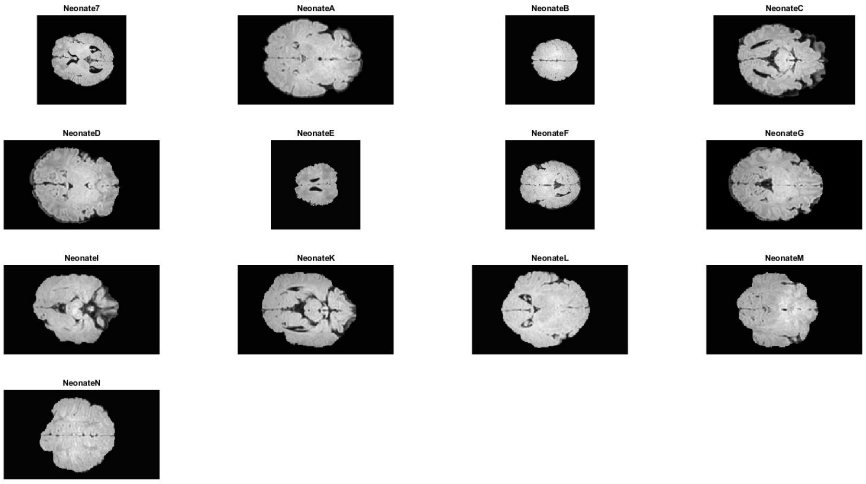


Figure 2.1: Display of the thirteen original patients' MRI (FLAIR mask)

2.2 MRI Acquisition

A crucial step in the outcome of this study was the image acquisition. In this case all the MR imaging was done using a 3 Tesla whole-body system (Ingenia 3.0T; Philips Healthcare, Best, Netherlands). After being fed and sedated the patients were attached a 32-channel head receiver array. During the scan, all the kids were laid in a supine position and an intensive care neonatologist or anaesthesiologist monitored their heart rate and oxygen saturation. The same standard clinical MR imaging protocol was used for all patients.

2.3 Clustering methods methods

In this thesis four different types of clustering segmentation were used on the MR images (together or separately) and were then compared to each other to assess their performance:

2.3.1 *k*-means algorithm

The k-means algorithm is a simple clustering procedure which is used to group different objects into clusters. An ideal cluster is a collection of data objects that has a minimum heterogeneity inside the array (intra-cluster variability) and a maximum distance between clusters (inter-cluster distance) [4]. The k-means is an unsupervised algorithm that in an iterative fashion obtains the clusters with the following steps:

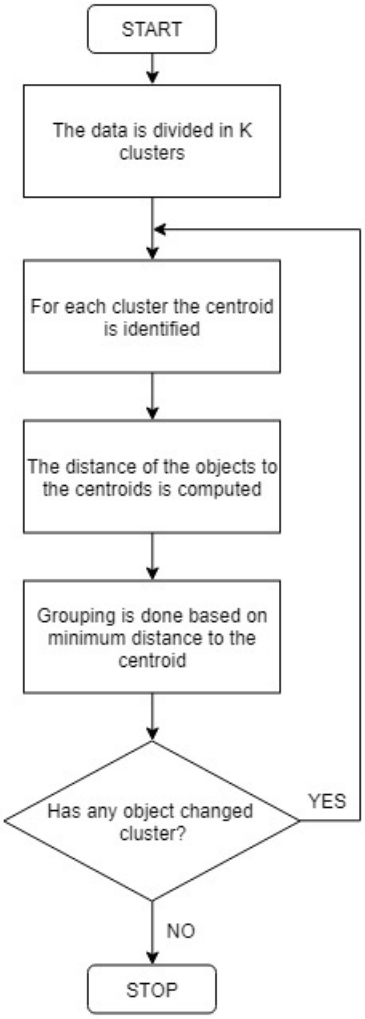


Figure 2.2: Steps of the k-means algorithm

So basically, after defining the number of clusters (k) the clusters are initialized by an arbitrary assignment of elements to clusters or a random set of centroids. Afterwards, the centroids of all the clusters are computed so all the elements can be reassigned, if needed, to the cluster with the nearest centroid, according to a specific distance measure. If there has been any reorganization of elements, the centroids are computed again, and all the distances are recalculated. However, if there is no restructuration, the clusters have been successfully found.

The main disadvantage of this algorithm is the need to previously set the number of clusters. Usually, properly setting this number previous to the clustering is not possible. In order to overcome is possible to:

1. apply the k-means algorithm starting from a defined set of clusters
2. post-processing the final clustering results, merging those clusters that are more similar
3. using an ISODATA algorithm

An ISODATA (Iterative self-organizing data analysis technique algorithm) is an extension of the k-means algorithm with some heuristics to automatically select the number of clusters. After selecting several required parameters, the algorithm works in an iterative fashion:

1. preforming the k-means clustering
2. splitting any clusters whose samples are sufficiently dissimilar
3. merging any two clusters closer enough
4. if clusters have not changed the iteration stops, whereas if there has been a change, the iteration is started again in the first step

2.3.2 Region-growing algorithm

There are different types of region-based clustering methods but all of them are based in partitioning a set of data in different areas using homogeneity criteria. This criterion is applied when going through the different items that may belong to a cluster and its proper adjustment has a direct effect on the obtained results. The characteristic that is used to select where the elements are arranged can be any property that can be computed such as colour, texture and/or intensity [5].

To initiate the region-based segmentation, a set of seed points needs to be selected. These can be selected by various ways and are the places where the initial regions begin. In the region-growing algorithm, the different areas are grown form said seed points according to the selected criteria.

The regions that want to be obtained using a clustering method cannot be disjoint since a pixel or item must not be classified in two different regions and all the items in the image need to be classified. In order to obtain the areas, the property selected for the homogeneity criterion must not be valid for any combination of

two or more areas. If this happens, it can be said that the images were properly segmented.

The region-growing methods have different ways of working, thee diagram below displays one possible sequence used to cluster all the items in an image:

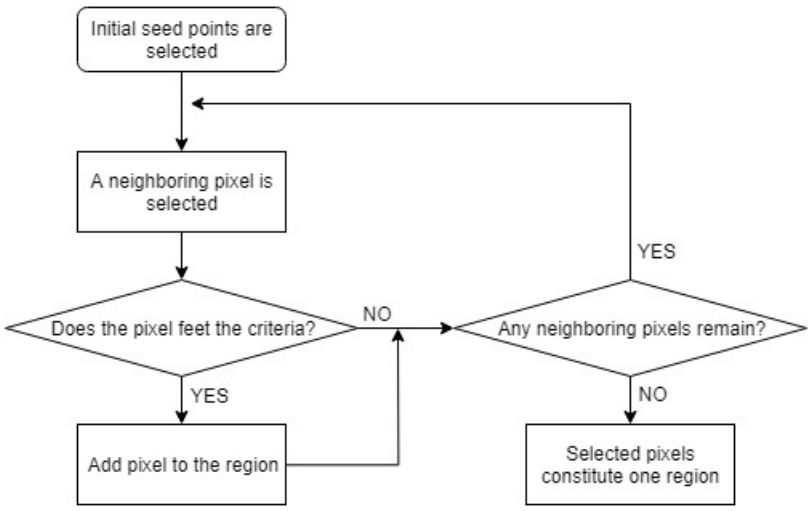


Figure 2.3: Steps of the region-growing algorithm

2.3.3 Hierarchical clustering

When segmenting an image there are two main iterative methods that can be used:

- Partitional or flat clustering algorithms: like the k-means algorithm, these algorithms produce a set of disjoint clusters.
- Hierarchical clustering algorithms: which produce a hierarchy of nested clusters.

Basically, hierarchical clustering groups data over a variety of scales by creating a cluster tree. This tree shows the union of sub clusters to achieve only one cluster for the image. Every intersection or node in the tree represents the different clusters and the hierarchy of clusters is known as dendrograms [6].

For instance, in a data group of n samples, the first partition consists of n cluster that each is constituted by exactly one sample. In the following partition two clusters are merged hence having *n-1* clusters now. There are as many partitions as clusters and in the last one all the samples form one cluster.

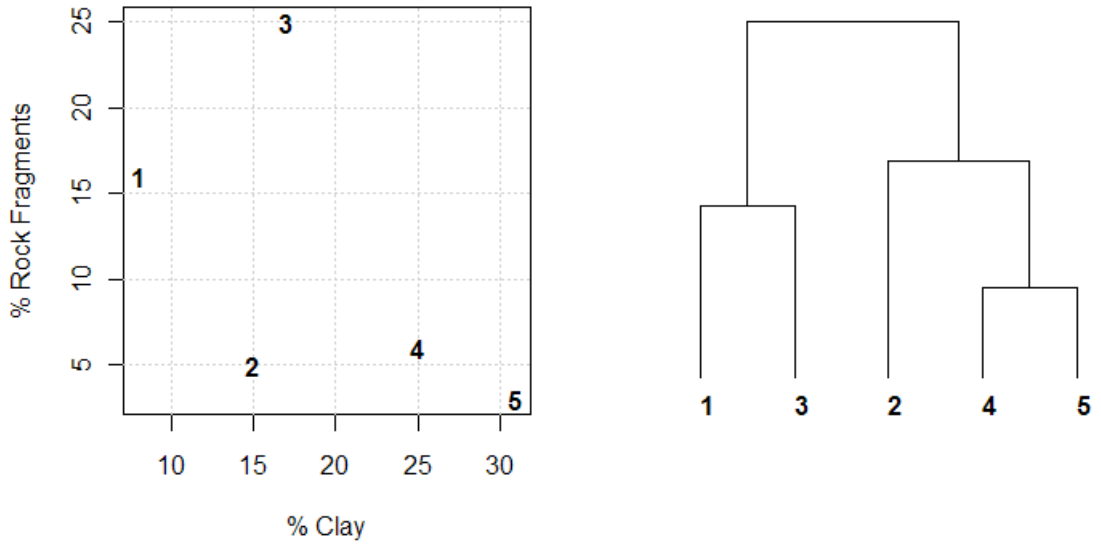


Figure 2.4: Example of a dendrogram

Given any two samples from a data group such as x and x', they will eventually be united in the same cluster. If these two samples remain in said cluster at higher levels, then the sequence is said to be a hierarchical clustering.

If the dendrograms are plotted is possible to measure the similarity between clusters and can be shown if the plot was drawn to scale. This value of similarity can be used to assess whether the clusters that were obtained are natural or forced. For instance, if these similarity values are evenly distributed throughout the possible values, there is no way to tell which specific number of clusters is better than another.

When dealing with the clusters there are two main methods: agglomerative (bottom-up) and divisive (top-down). Whether the clusters are combined or split depends on a measure of dissimilarity between sets of observations which is assessed with a measure of distance between pairs and the degree of unlikeness of sets.

2.3.4 Statistical parametric mapping (SPM)

To obtain the standard masks, which were used to assess the previously mentioned clustering methods, a software module called Statistical Parametric Mapping (SPM) was used. This method is based on a modified Gaussian mixture model (Ashburner & Friston 2000).

In this software the Bayesian rule is used to assign a probability for each voxel which assesses whether it belongs to each tissue class based on combining the

possibility of it belonging to that tissue class and the prior probability obtained from previous probability maps which were computed for a large number of subjects.

The main advantage that using this data analysis tool provides is that it objectively interprets the images and it is reliable. For this reason the standard segmentation was done with this method.

2.4 Evaluation methods

The main objective of this thesis is to evaluate the different methods to try and find a better brain segmentation, to do so appropriate assessment methods were needed. After careful consideration, the following methods were used:

2.4.1 Sørensen-Dice index

The Sørensen-Dice index is a similarity coefficient for image segmentation that assesses the resemblance between two images by assigning a similarity index [7]. This index is a number between zero and one, where one represents two identic images and zero stands for two images that are opposites of each other. For two data sets A and B, the dice index is calculated as follows:

$$D = 2 \times \frac{(A \cap B)}{|A| + |B|} \tag{2.1}$$

This index is widely used to compare the similarity between two images because it is extremely robust.

To compare the standard masks with the ones obtained with the clustering methods, the dice index was computed for different binarization thresholds and for each slice.

2.4.2 Pixel classification

This assessment consists on using the definition of true positive (TP), false positive (FP), false negative (FN) and true negative (TN) to determine whether the pixels are being properly classified into the corresponding regions. A true positive is an outcome where the model correctly predicts the positive class. Similarly, a true negative is an outcome where the model correctly predicts the negative class. A false positive is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class.

Table 2.1: Pixel classification

	TP	FN	FP	TN
Recognised in the standard	YES	YES	NO	NO
Recognised in the clustering	YES	NO	YES	NO

When comparing the SPM segmentation with the clustering, the pixel classification was done for each slice and for each row in a single slice.

2.4.3 Visual analysis

After evaluating a few methods that had been applied to the thirteen initial neonates and obtaining unexpected results from some of the test subjects, the standard masks were visually analysed. This was done by plotting several slices for each patient and visually assessing whether this segmentation was good enough to be considered a standard.

For example, NeonateB was discarded because most of the pixels had been classified as white matter and the grey matter was almost empty and, since that is physically impossible, it was deemed as an incorrect segmentation.

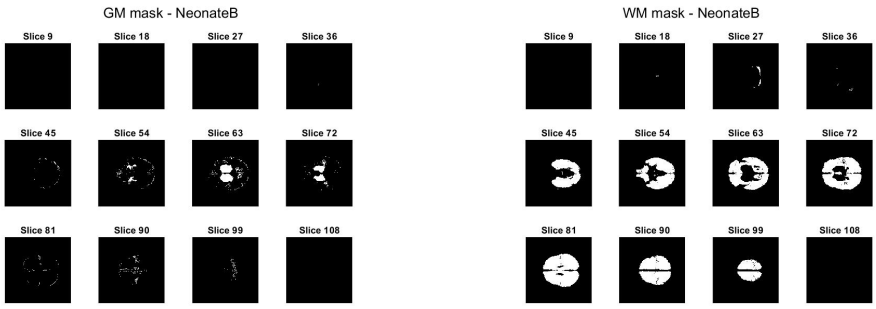


Figure 2.5: NeonateB's standard masks

Since this type of assessment is not being completely objective and could lead to erroneous conclusions only those neonates whose masks were completely wrong segmented were removed from the study group. The reason for this removal was to avoid making modifications on the code based on erroneous results.

Due to this being done after having already obtained some results, in the first part of the thesis there are thirteen test subjects while the last part only has eight tests subjects.

After this assessment the test subjects were as follows:

Table 2.2: Test subject selection

Patient	Test subject
Neonate 7	Yes
Neonate A	Yes
Neonate B	No
Neonate C	Yes
Neonate D	Yes
Neonate E	Yes
Neonate F	Yes
Neonate G	Yes
Neonate I	No
Neonate K	No
Neonate L	No
Neonate M	No
Neonate N	No

2.5 Initial code

This thesis is the continuation of a previous study made by S. Rosati, B. Toselli, M.M. Fato, D. Tortora, M. Severino, A. Rossi and G. Balestra. From their work four clustering versions were obtained: v0, v0.1, v2 and v2.2. The four versions work in an extremely similar fashion and all obtain three masks (GM, WM and CSF) as a results of each test subject's MRI segmentation.

The code consists on twelve different Matlab files, eleven of them being functions. Each plays an important role when obtaining the segmented masks.(All these functions can be seen in the Appendices)

- *main_perArticolo*: Matlab script where the T1, T2 and FLAIR masks are loaded segmented and a final mask is obtained with all three segmentations (GM, WM and CSF). [Appendix A]
- *im_reorientation*: function used in the *main_perArticolo* that rotates the volumetric image from the sagittal plane to the axial plane if the matrix dimensions of the T1, T2 and FLAIR masks are not the same. [Appendix B]
- *calcolo_maschera_CSF1*: function that using the T1 and FLAIR masks computes the CSF segmented mask.[Appendix C]
- *calcolo_maschera_GM*: function that using the T1, FLAIR and CSF masks computes the GM segmented mask.[Appendix D]
- *calcolo_maschera_WM*: function that using the T1, FLAIR and CSF+GM masks computes the WM segmented mask.[Appendix E]

- *creazione_matr_clustering*: function that creates the matrix to be used for clustering from a series of images. [Appendix F]

- *region_growing1*: function that applies the region-growing clustering method when obtaining the WM mask and the CSF mask. [Appendix G]

- *region_growing*: function that applies the region-growing clustering method when obtaining the WM mask with different criteria than *region_growing1*. [Appendix G]

- *scarta_pixel*: function that discards the pixels that have been included in previous masks. [Appendix I]

- *calcolo_medie*: function that computes the average of all the pixel values in the clusters that have been obtained. [Appendix J]

- *creazione_maschere*: function that creates different masks for each of the identified clusters so they can be plotted separately. [Appendix K]

- *clust_image*: function that applies the k-means algorithm to obtain a determinate amount of clusters from a slice, if the clusters are too small (number of elements <5% total elements) the amount of clusters is increased by one. [Appendix L]

All these files are related as follows:

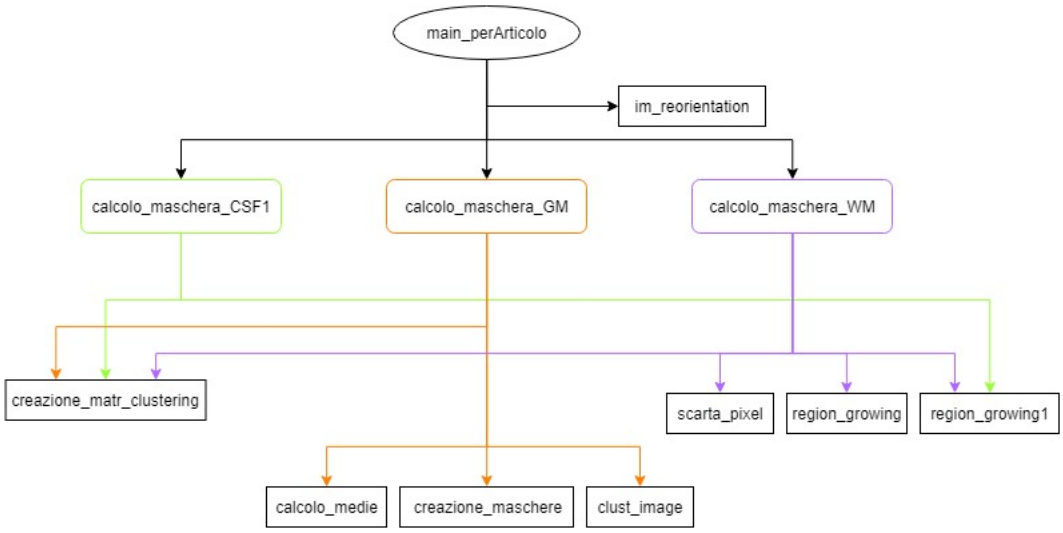


Figure 2.6: Original version: code files

Basically, the *af*BTS algorithm works on axial T1, T2 and FLAIR sequences and obtains the segmented masks in four steps:

1. *Skull-stripping and denoising*

   To begin with, all the extra-cerebral tissues are removed. This includes the skull, eyeballs and skin and it is done with the Brainsuite software (v. 16a). What this software does is, using the Brain Surface Extractor (BSE) with specific parameters, compute a binary mask using the T1 sequence for each test subject. Both the T2 and the FLAIR sequences are multiplied with the previously obtained brain mask to be masked.

   Afterwards, the T1 and FLAIR are processed with an average filter (low pass) filter, which is a 3-by-3 kernel, to remove any noise that can be found in the masks. Then all the intensities which range 0 to 255 are normalized between 0 and 1. For each patient this is done for each slice of the three sequences.

2. *CSF segmentation*

   In this step the T2 sequence is not used. To begin with, each slice from the T1 sequence is multiplied with the correspondent FLAIR slice to compute a new sequence called *T1*FLAIR*.

   In the next figure three slices from one of the neonates can be seen. Panels A and B show the T1 and FLAIR slices, respectively. In panel C an example of a *T1*FLAIR* slice obtained from the other two sequences can be seen. In this last panel it is noticeable how the CSF appears darker which makes for an easier identification.

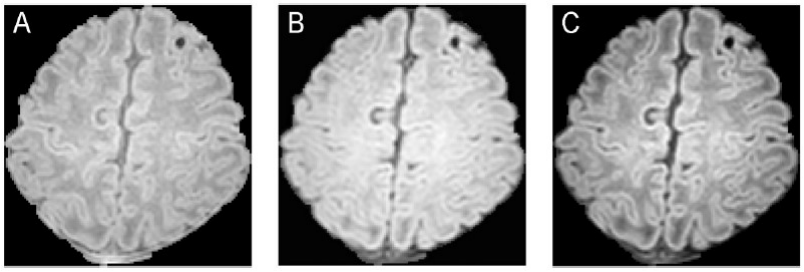


Figure 2.7: Example of a T1 (A), FLAIR (B) AND *T1*FLAIR* (C) of a neonate

   To this new sequence a region growing (RG) algorithm is applied. In this code the seed points are identified for each slice as those voxels with an intensity lower than the 10^{th} percentile of all the intensities in the slice. The similarity is set to the 15^{th} percentile of the intensities. Both of the binary masks that are obtained with this algorithm are summed and scaled between 0 and 1.

   By this procedure a binary mask the CSF_i is obtained. This mask represents the cerebrospinal fluid of the i-th slice. Since this masks usually have discontinuities, a final mask of the i-th slice ($\text{CSF}_{i,tot}$) is calculated as the weighted sum of the current slice and the previous and following slices:

$$CSF_{i,tot} = 0.3 \cdot CSF_{i-1} + 0.4 \cdot CSF_i + 0.3 \cdot CSF_{i+1} \tag{2.2}$$

The masks obtained with this equation assure the anatomical continuity of the obtained segmentation across all slices. Each voxel has assigned a value between 0 and 1 which represents the probability of said voxel to be part of the CSF. Lastly, to make the mask clearer, all the voxels with a probability lower than 0.4 are removed along with any connected region which has less than four voxels. This holes with less than four voxels are filled in.

3. *GM segmentation*

   In this step the FLAIR sequence is not used. Firstly, all the voxels that were assigned to the CSF mask are removed from the T1 and T2 sequences, this is done so the remaining voxels can only belong to the WM or GM mask.

   To start, a new sequence called $T2^c*T1$ is obtained by complementing all the slices in T2 with respect to 1 and then multiplying them by their correspondent slices of the T1 sequence. In the next figure three slices from one of the neonates can be seen. Panels A and B show the T1 and T2 slices, respectively. In panel C an example of a $T2^c*T1$ slice obtained from the other two sequences can be seen. In this last panel it is noticeable how the GM appears lighter which makes for an easier identification.

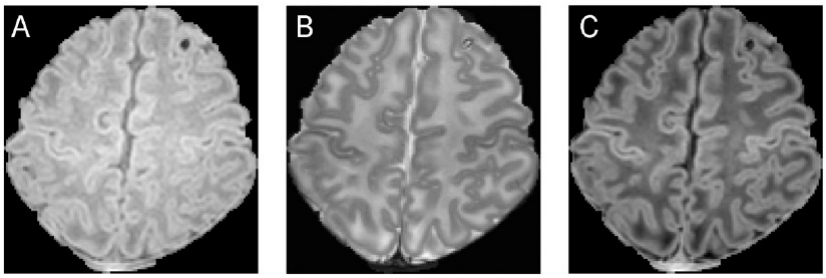


Figure 2.8: Example of a T1 (A), T2 (B) AND $T2^c*T1$ (C) of a neonate

Then, to be able to consider the variation and gradualness of the myelination process in the neonates, each slice is divided into different areas. For the same reason, a k-means algorithm is applied with a number of clusters (k) that progressively increases from 3 to 9 in a caudal direction. Slice-by-slice, the k-means is applied to the T2 sequence's voxels (considering only GM and WM voxels). This way, k areas are obtained from each slice that have similar intensities in all of their voxels. On each of the correspondent $T2^c*T1$ slices, these areas identified on the T2 slices are mirrored.

Further application of the k-means algorithm, with k equal to two, to the T2 intensities for each area results in a better separation of GM and WM voxels. For every area, all the voxels that belong to the cluster which has the highest mean intensity are assigned to the GM mask, which is a binary mask. These steps are repeated on the $T2^c*T1$ sequence to obtain another GM mask that, similar to the CSF segmentation, is added to the previous GM mask obtained. This addition is then done to assure the anatomical continuity of the segmented regions:

$$GM_{i,tot} = 0.3 \cdot GM_{i-1} + 0.4 \cdot GM_i + 0.3 \cdot GM_{i+1} \tag{2.3}$$

The final mask obtained contains the probability, between 0 and 1, of every voxel to belong to the GM mask. Finally, a post-processing of the mask is done by removing any voxel with a probability lower than 0.5. As far as small regions goes, any region with an area smaller than four voxels are removed while holes of the same size are filled in.

4. *WM segmentation*

   Finally, to obtain the WM mask, all those voxels not assigned to any of the previous masks (CSF and GM) are further processed along with the T1 and T2 sequences. To start with, for the remaining voxels, the $T2^c * T1$ is computed. After, an RG algorithm is applied to both the T1 and $T2^c * T1$ sequences, independently. For the T2 sequence, the seed points are selected as those voxels with intensities above the 50^{th} percentile of all the intensities in every slice and the similarity threshold is the 5^{th} percentile of intensities. For each slice in the $T2^c * T1$ sequence the voxels with intensities below the 20^{th} percentile are used as seed points while the 90^{th} percentile is set as a similarity threshold.

   After applying the RG algorithm, two binary masks are obtained for all the slices. These masks are summed and then scaled between 0 and 1. The final mask for the i-th slice is the result of the following weighted sum:

$$WM_{i,tot} = 0.3 \cdot WM_{i-1} + 0.4 \cdot WM_i + 0.3 \cdot WM_{i+1} \tag{2.4}$$

   Finally, all voxels with intensities lower than 0.4 are deleted, regions with less than four voxels are also deleted and holes smaller than four voxels are filled in.

Chapter 3

Implementation

3.1 Initial code evaluation

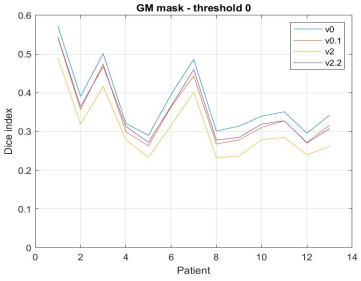
The first step in this project was asses how accurate the initial code versions were. To do that, the segmentations obtained using the initial clustering codes were compared to the masks obtained by SPM segmentation. So, for each of the seven patients' MRI, the grey matter (GM) mask and the white matter (WM) standard masks were compared to the corresponding masks obtained with four different versions of the clustering code (v0, v0.1, v2, v2.2). These masks were all converted to the 0 to 1 scale and then binarized.

Using Matlab, the dice indices were calculated for each slice and an average value was obtained for each mask. The indices were estimated using one binarizing threshold for all patients and using multiple thresholds for each individual patient. For each threshold it can be seen that the different patients have different dice indices depending on which version of the clustering code was used to obtain the masks.

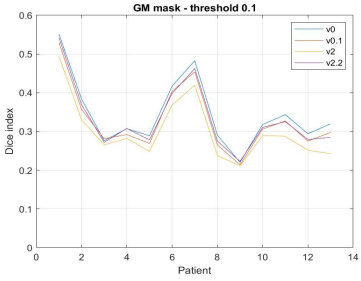
3.1.1 Results for different thresholds

Firstly, all the different versions were analysed using multiple binarizing threshold. These thresholds are used to binarize both the SPM and the clustering masks so that computing the dice index is possible. The goal of doing this is finding out which binarizing threshold is better so that in future version only one threshold is used. This will speed up the process of analysing the results to hopefully find a better segmentation version.

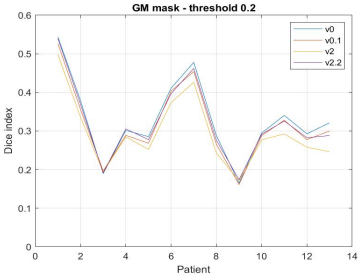
In the following images what can be seen is the dice indices of the thirteen neonates separated in their WM and GM masks and with different thresholds ranging from 0 to 1. Note that the last threshold is 0.9 and not 1 because when binarizing with threshold 1 the dice index is 0 so no information can be obtained from this.



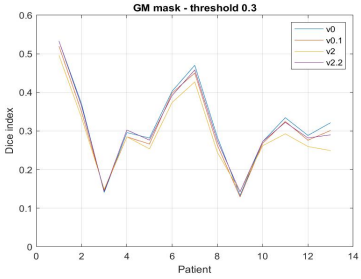
(a) GM mask - threshold 0



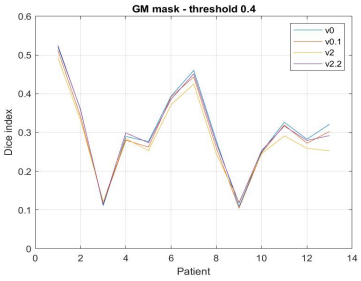
(b) GM mask - threshold 0.1



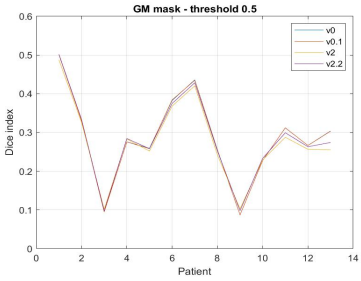
(c) GM mask - threshold 0.2



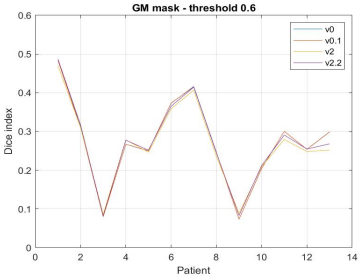
(d) GM mask - threshold 0.3



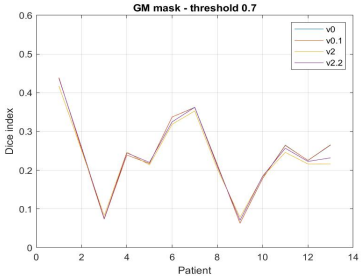
(e) GM mask - threshold 0.4



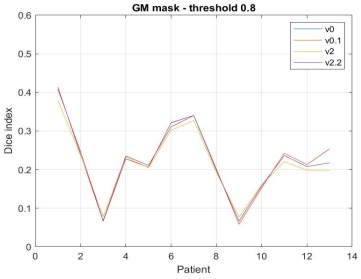
(f) GM mask - threshold 0.5



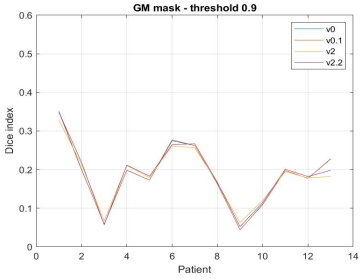
(g) GM mask - threshold 0.6



(h) GM mask - threshold 0.7

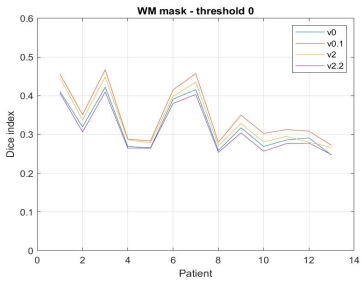


(i) GM mask - threshold 0.8

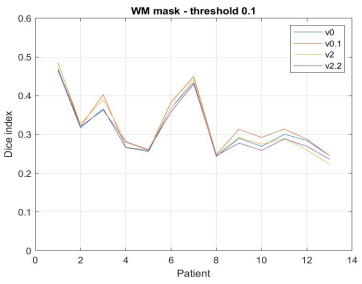


(j) GM mask - threshold 0.9

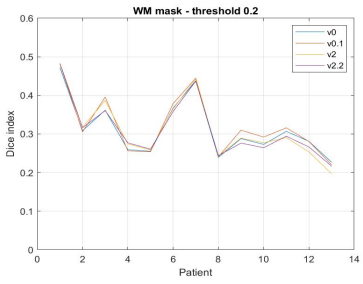
Figure 3.1: Dice indices for the GM mask with different binarizing thresholds



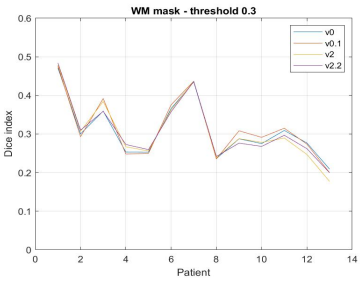
(a) WM mask - threshold 0



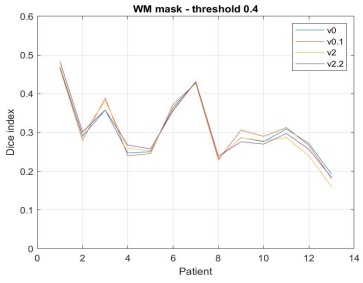
(b) WM mask - threshold 0.1



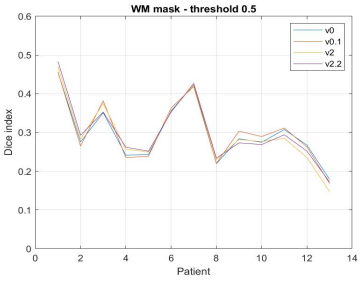
(c) WM mask - threshold 0.2



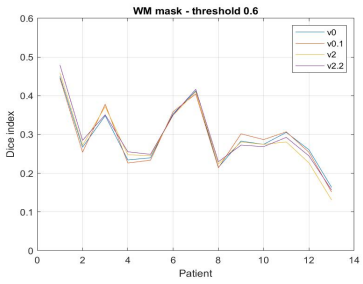
(d) WM mask - threshold 0.3



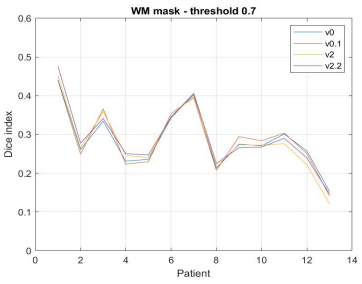
(e) WM mask - threshold 0.4



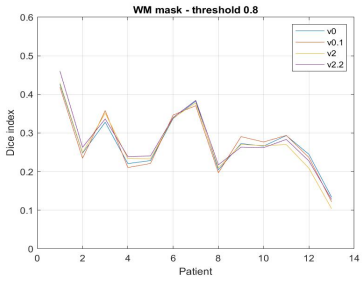
(f) WM mask - threshold 0.5



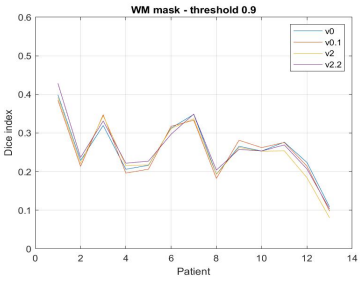
(g) WM mask - threshold 0.6



(h) WM mask - threshold 0.7



(i) WM mask - threshold 0.8



(j) WM mask - threshold 0.9

Figure 3.2: Dice indices for the WM mask with different binarizing thresholds

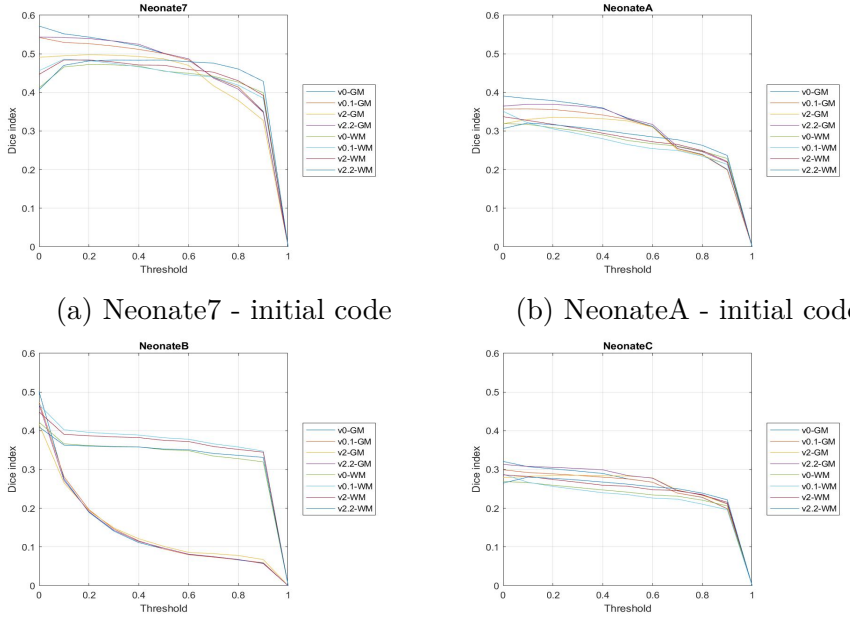
With the obtained results overall better results are clearly achieved when using 0.4 as the binarizing threshold for both the WM and the GM masks. So, from now on, all the obtained masks are going to be binarized with said threshold before computing the dice index.

It should also be noted that NeonateB has lower indices in the GM masks for any threshold higher than 0.1. This is going to be addressed in further versions if it does not improve with the other versions.

3.1.2 Results for different versions

The next step was plotting the indices for all the clustering versions for each patient. This was done so that it could be seen with which version higher indices could be obtained and to reassure that the binarizing threshold found in the previous results was correct.

In the following images a plot for each patient is displayed. In every plot the dice indexes obtained are represented for different binarizing thresholds ranging 0 to 1. The different lines correspond to the four clustering versions for the GM and the WM masks. In these images the threshold 1 was plotted for visual aid but still no further information can be obtained from it.



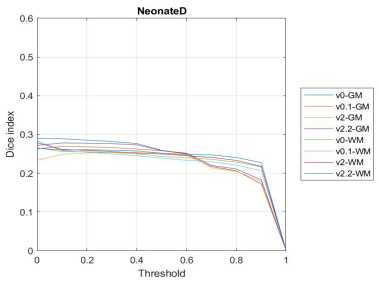
(a) Neonate7 - initial code

(b) NeonateA - initial code

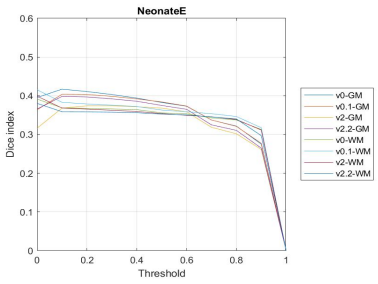
(c) NeonateB - initial code

(d) NeonateC - initial code

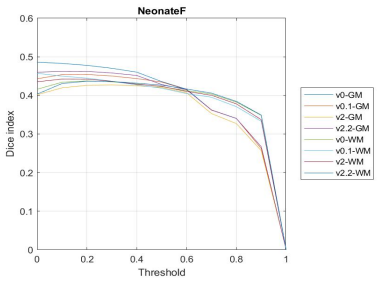
Figure 3.3: Dice indices for initial code versions



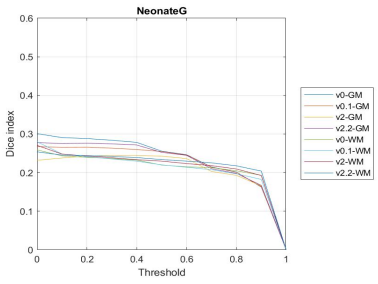
(e) NeonateD - initial code



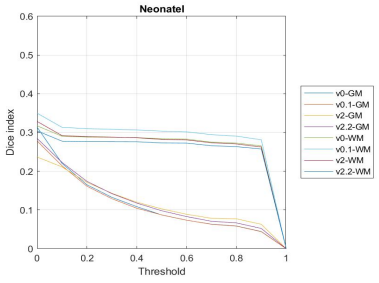
(f) NeonateE - initial code



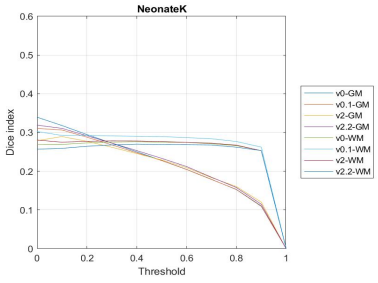
(g) NeonateF - initial code



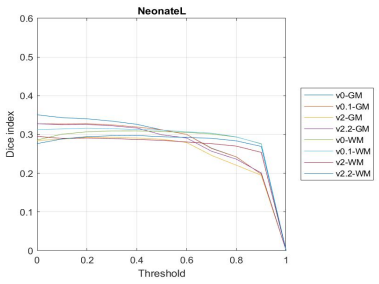
(h) NeonateG - initial code



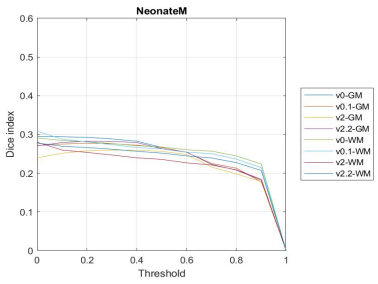
(i) NeonateI - initial code



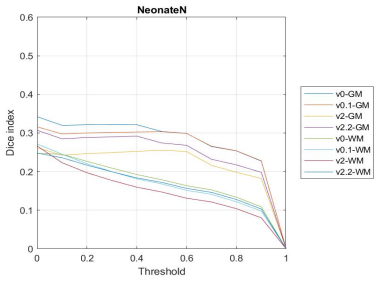
(j) NeonateK - initial code



(k) NeonateL - initial code



(l) NeonateM - initial code



(m) NeonateN - initial code

Figure 3.3: Dice indices for initial code versions

Even though not all test subjects act the same way, the conclusions are extracted based on the most common behaviour that can be found in the previous images. So basically, with the obtained results it can be seen that the higher indices are found when the version 0 is used to obtain the grey matter mask while for the white matter mask the version 0.1 achieves a better segmentation. Furthermore, it is confirmed that superior results are obtained binarizing the masks at 0.4.

3.2 Code combination

Based on the obtained results a new code version was made combining the best initial code versions as follows:

Table 3.1: Code files for v3

Function	Version
main_perArticolo	v3
im_reorientation	v0
calcolo_maschera_CSF1	v0
calcolo_maschera_GM	v0
calcolo_maschera_WM	v0.1
creazione_matr_clustering	v0
region_growing1	v0.1
region_growing	v0.1
scarta_pixel	v0.1
calcolo_medie	v0
creazione_maschere	v0
clust_image	v0

So essentially, the new code version (v3) was obtained from the combination of v0 and v0.1. The grey mask is computed using v0 and the white mask using the v0.1. The segmentation obtained using the clustering method v3 was then compared to a segmentation obtained by SPM segmentation. The computed dice indices showed better results than those obtained for the previous versions.

To further evaluate how accurate the new version was different evaluation methods were used:

3.2.1 Dice index for each slice

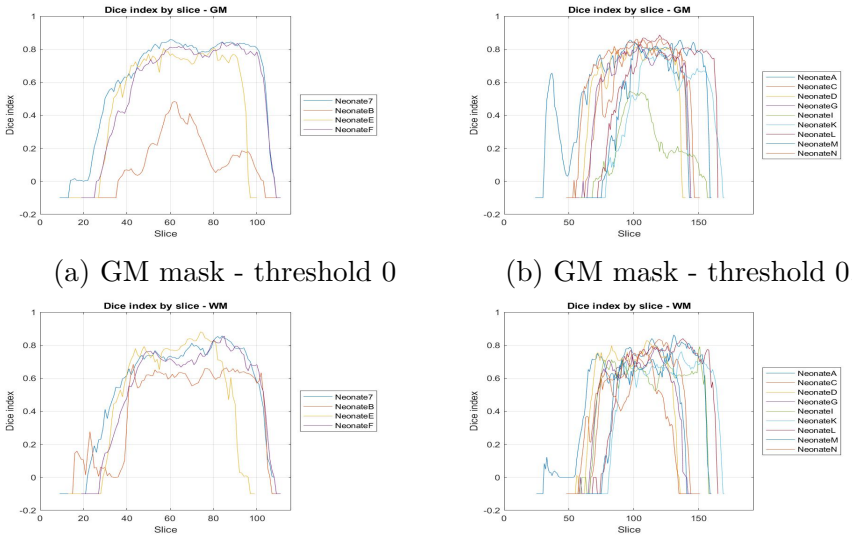
After binarizing all images using the 0.4 binarizing threshold, all the obtained segmentations were analysed in depth. The first step was analysing how the dice index changes throughout the slices of each mask. So, for each patient, the dice indices were computed again for each slices of their third dimension.

In the following images the dices indices for each slice can be seen. Since the patients have different number of slices so the same results can be seen in two different plots according to the number of slices.

As to represent the non-positive values obtained when computing the dice index the following values were introduced:

- $-0.1 \rightarrow$ slice has tissue not recognised by the standard but recognised by the algorithm
- $-0.2 \rightarrow$ slice has tissue recognised by the standard but not recognised by the algorithm
- $0 \rightarrow$ slice has tissue recognised both by the standard and the algorithm but they have no intersection

These were obtained analysing the slices where the dice index computed was 0.



(a) GM mask - threshold 0

(b) GM mask - threshold 0.1

(c) GM mask - threshold 0

(d) GM mask - threshold 0.1

Figure 3.4: Dice indices for each slice in both GM and WM masks

In these four images it can be seen how the first and last slices are where the lowest indices can be found in all the neonates. This does not depend on whether

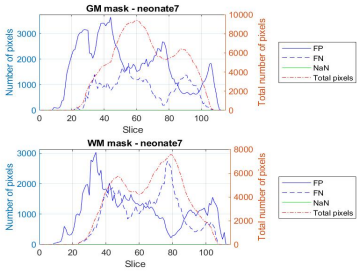
the masks have more or less slices in their third dimension. In some cases, this accounts for the low dice index since the middle indices are not so low.

3.2.2 Tissue recognition by slice

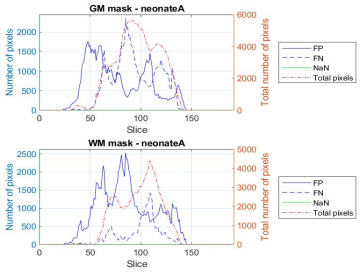
Since the first and last slices proved to be the ones that were being poorly segmented, a new analysis was done to see which kind of mistake the code was making. To do so the image was checked pixel by pixel to see where the segmentation was not being done properly and why.

For each test subject, the pixels were divided into five categories based on Table 2.1: true positive, true negative, false positive, false negative and NaN (Not a number).

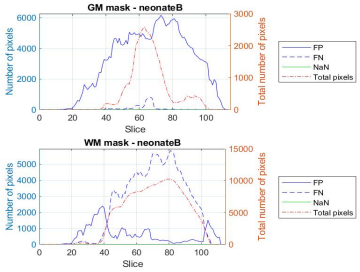
Essentially, for each test subject, both GM and WM masks were looked at pixel by pixel and compared to the same pixel in the SPM mask. This comparison allowed for each pixel to be classified in one of the five categories. Then the results were plotted so for each slice it could be seen how many pixels belonged to each category. But, since the number of false positives correlates with the number of true positives and the number of false negatives correlates with the number of true negatives, only the false classifications were shown in the final plots.



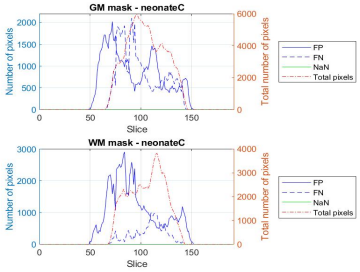
(a) Neonate7 - pixels by slice



(b) NeonateA - pixels by slice

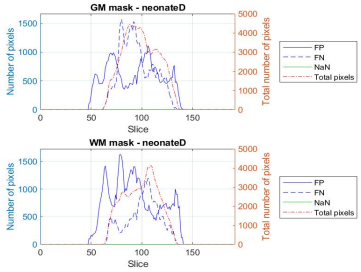


(c) NeonateB - pixels by slice

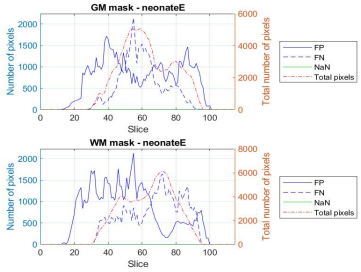


(d) NeonateC - pixels by slice

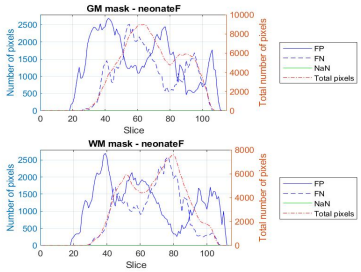
Figure 3.5: Pixel classification by slice for v3



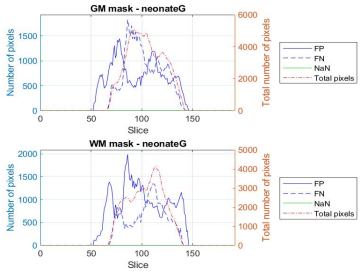
(e) NeonateD - pixels by slice



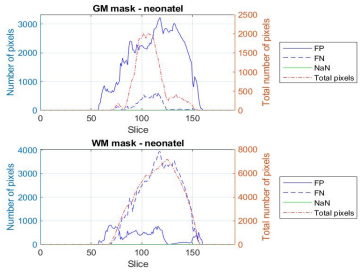
(f) NeonateE - pixels by slice



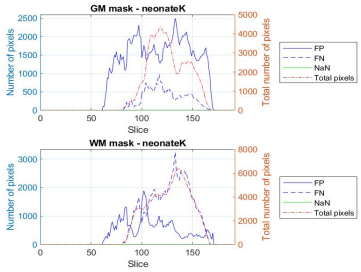
(g) NeonateF - pixels by slice



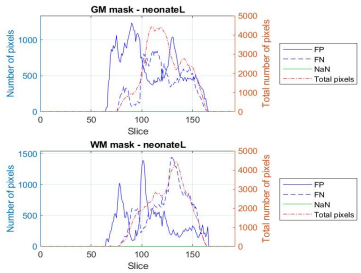
(h) NeonateG - pixels by slice



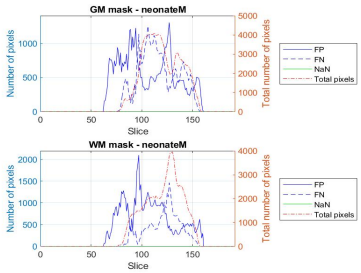
(i) NeonateI - pixels by slice



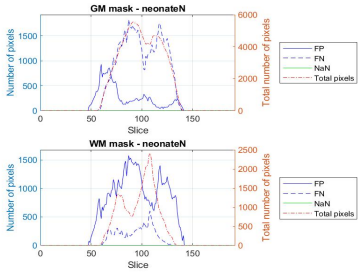
(j) NeonateK - pixels by slice



(k) NeonateL - pixels by slice



(l) NeonateM - pixels by slice



(m) NeonateN - pixels by slice

Figure 3.5: Pixel classification by slice for v3

It can be seen that, when not correctly classifying a pixel most of the times is by a false positive rather than a false negative. So, areas which should not be considered in a mask are being classified as part of the mask.

In some case it can be seen that the false negatives are higher than the false positives. This clearly happens in: NeonateB's white matter mask, NeonateI's white matter mask, NeonateK's white matter mask and NeonateN's grey matter mask.

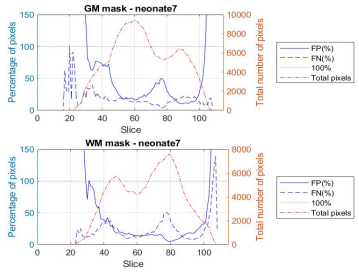
3.2.3 Tissue recognition by slice - percentage

After find out how the pixels from each slice were being incorrectly classified, new plots were made to see what percentage of the total pixels every classification accounted for. This was done using the following equations for each slice:

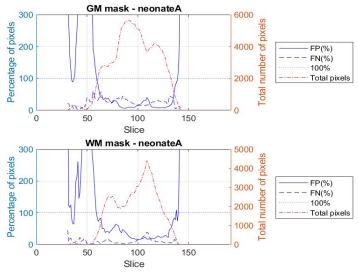
$$FP(\%) = \frac{total\ FP\ in\ v3}{total\ FP\ in\ SPM} \cdot 100 \tag{3.1}$$

$$FN(\%) = \frac{total\ FN\ in\ v3}{total\ FN\ in\ SPM} \cdot 100 \tag{3.2}$$

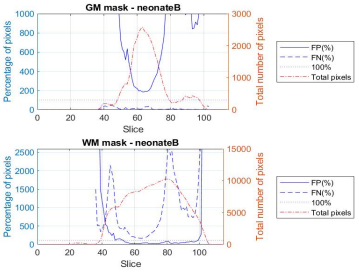
The 100% of the pixels in the SPM is computed for each specific slice before computing the percentages.



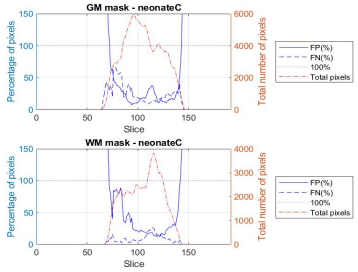
(a) Neonate7 - pixels by slice(%)



(b) NeonateA - pixels by slice(%)

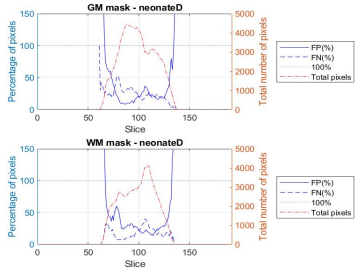


(c) NeonateB - pixels by slice(%)

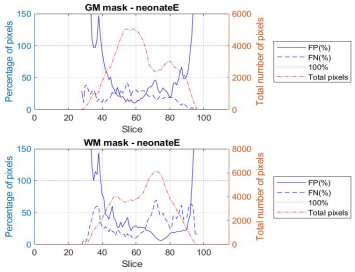


(d) NeonateC - pixels by slice(%)

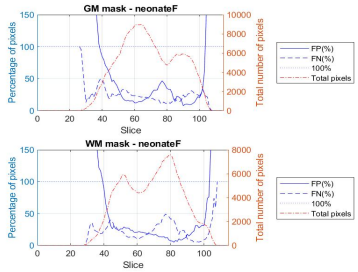
Figure 3.6: Pixel classification by slice for v3 (percentage)



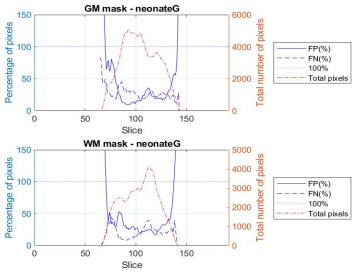
(e) NeonateD - pixels by slice(%)



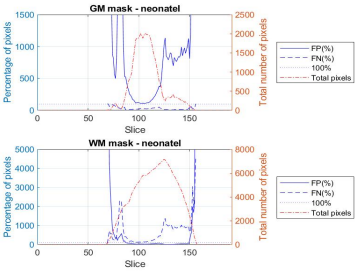
(f) NeonateE - pixels by slice(%)



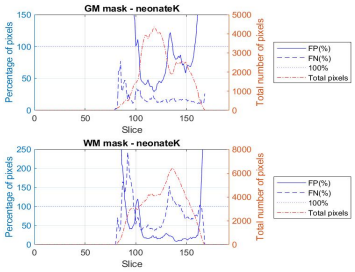
(g) NeonateF - pixels by slice(%)



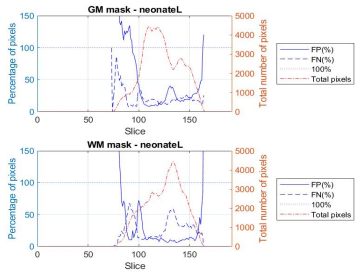
(h) NeonateG - pixels by slice(%)



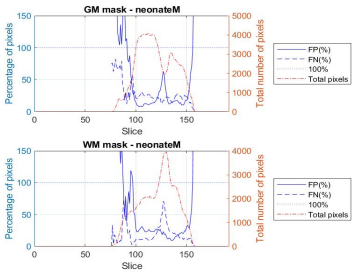
(i) NeonateI - pixels by slice(%)



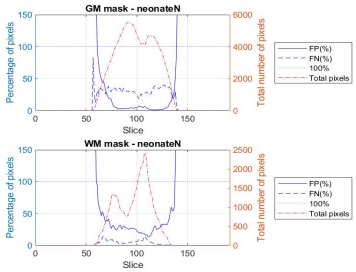
(j) NeonateK - pixels by slice(%)



(k) NeonateL - pixels by slice(%)



(l) NeonateM - pixels by slice(%)



(m) NeonateN - pixels by slice(%)

Figure 3.6: Pixel classification by slice for v3 (percentage)

When representing the percentage of false positives and false negatives it can be seen that the first and last slices have the most error. This is because a lot of pixels are getting recognised by our code while they are not recognised by the standard.

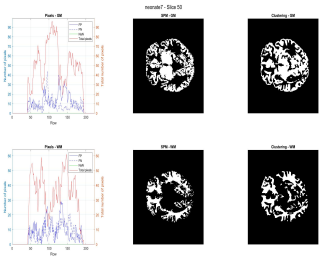
In Neonate B, I and K the number of false positives and negatives is mostly over 100% of the pixels and by a lot.

3.2.4 Tissue recognition by row

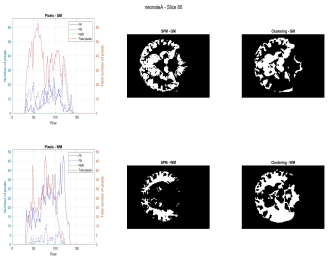
Once the behaviour of the pixel classification had been found for every slice, one slice of each patient was selected and analysed row by row. This analysis was done in the same way as the previous obtained results but instead of analysing slice by slice, a specific slice was selected and analysed row by row.

Besides plotting the obtained results, the selected slice was displayed for the standard masks and the clustering masks.

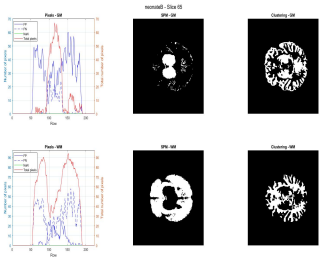
Similarly to the obtained results in the slice by slice analysis, looking at the plotted results it can be seen that false positives account for most of the pixel's misclassification. The first and last rows, which account for the most misclassifications, are those which are mainly empty.



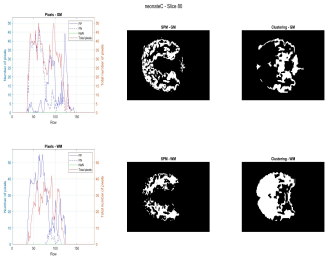
(a) Neonate7 - pixels by row



(b) NeonateA - pixels by row

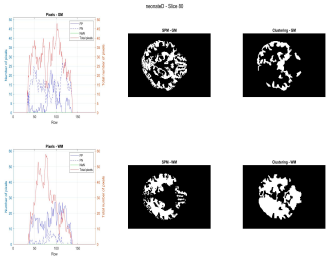


(c) NeonateB - pixels by row

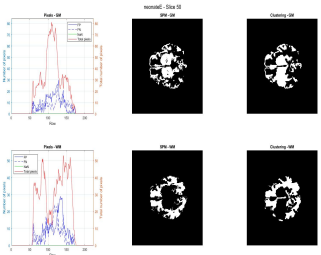


(d) NeonateC - pixels by row

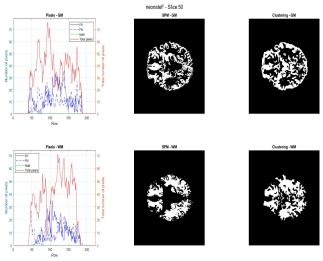
Figure 3.7: Pixel classification by row for v3



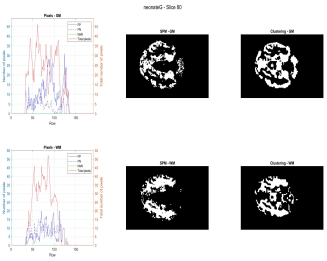
(e) NeonateD - pixels by row



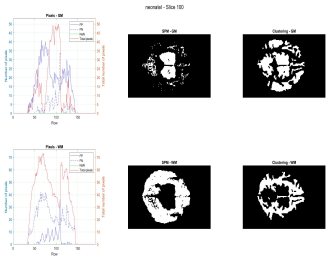
(f) NeonateE - pixels by row



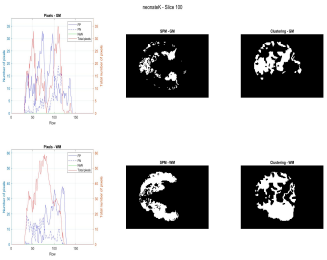
(g) NeonateF - pixels by row



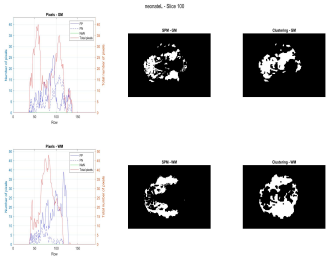
(h) NeonateG - pixels by row



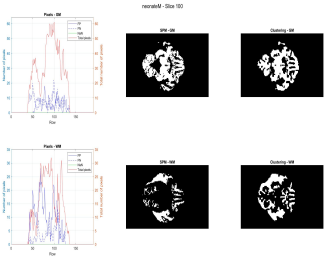
(i) NeonateI - pixels by row



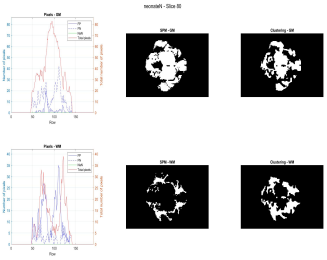
(j) NeonateK - pixels by row



(k) NeonateL - pixels by row



(l) NeonateM - pixels by row



(m) NeonateN - pixels by row

Figure 3.7: Pixel classification by row for v3

3.3 Threshold modification

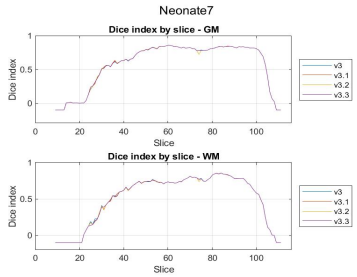
In each code version, when obtaining the three masks a threshold is used. In all the previous code versions these thresholds had a value of 0. To see how the modification of this threshold affected the segmentation result, the threshold was changed to 0.4 and the dice indices were computed to evaluate the results.

Three new code versions were obtained as follows:

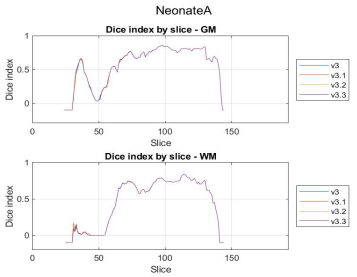
Table 3.2: Code files for v3

Version	GM threshold	WM threshold
v3.1	0.4	0
v3.2	0	0.4
v3.3	0.4	0.4

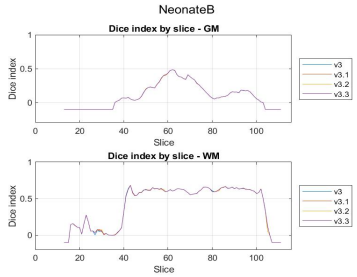
The obtained results where compared to the SPM segmentation and the dice index for each slice was obtained with a 0.4 binarization threshold. In the following plots the values that represent the non-positive values obtained when computing the dice index are as previously described in page 23.



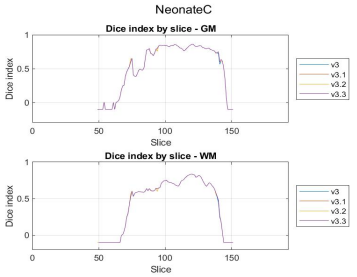
(a) Neonate7 - v3 modified



(b) NeonateA - v3 modified

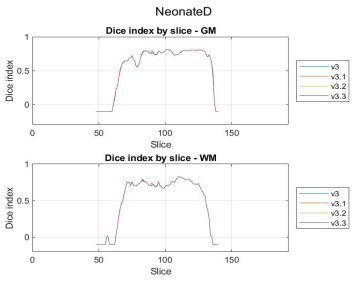


(c) NeonateB - v3 modified

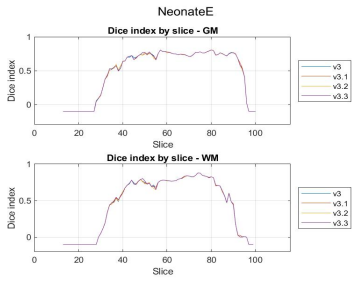


(d) NeonateC - v3 modified

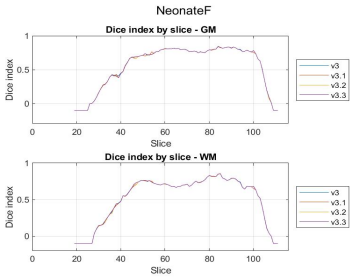
Figure 3.8: Threshold modification results



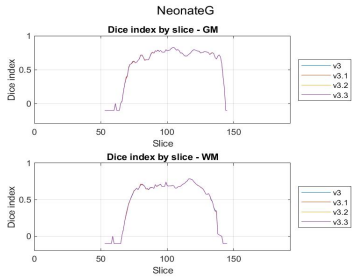
(e) NeonateD - v3 modified



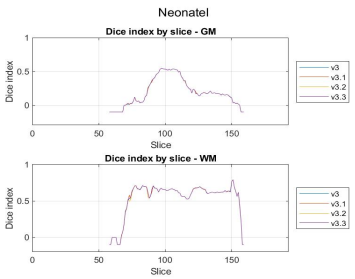
(f) NeonateE - v3 modified



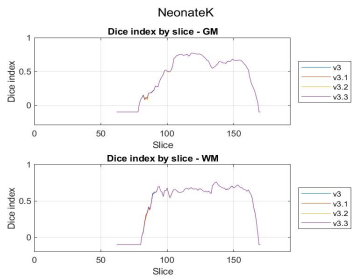
(g) NeonateF - v3 modified



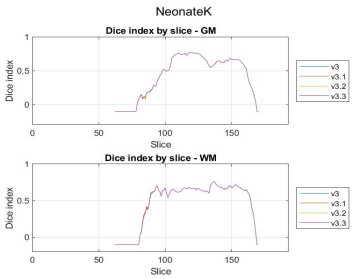
(h) NeonateG - v3 modified



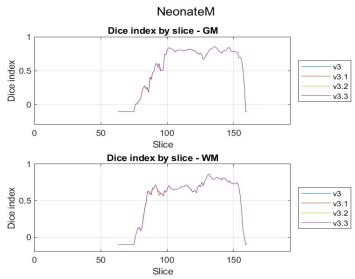
(i) NeonateI - v3 modified



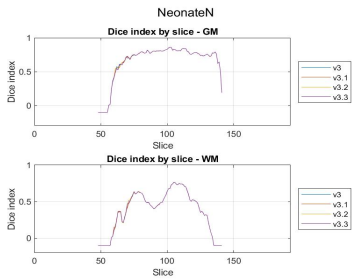
(j) NeonateK - v3 modified



(k) NeonateL - v3 modified



(l) NeonateM - v3 modified



(m) NeonateN - v3 modified

Figure 3.8: Threshold modification results

No significant change was obtained from the modification of the thresholds used to get the masks.

Even when evaluating slice by slice the results only vary in certain points sometimes for the better sometimes for the worse in the same mask but neither change is significant.

3.4 Hierarchical clustering

A new code version (v4) was written containing a 0.4 threshold for obtaining the masks (v3.3) and instead of a k-means method, the GM mask was obtained by different methods of hierarchical clustering. The different methods included two metrics: Euclidean distance and "cityblock" distance; and two linkage criteria: centroid linkage clustering and maximum or complete-linkage clustering.

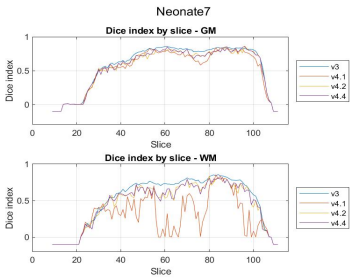
The new versions were as follows:

Table 3.3: Hierarchical clustering versions

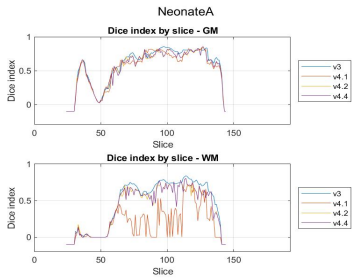
Version	Method	Distance
v4.1	centroid	Euclidean
v4.2	complete	Euclidean
v4.3	centroid	"cityblock"
v4.4	complete	"cityblock"

Since the centroid distance is only appropriate for Euclidean distances, v4.3 was discarded after no good results were obtained from it.

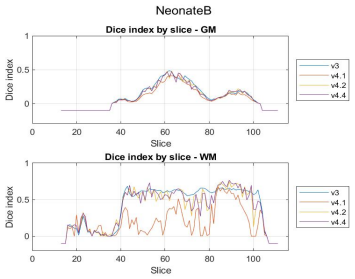
After the masks were obtained, the dice indices were computed slice by slice for all the test subjects. In the plots, v3 was also displayed since, from the previous versions, it was the one with the highest dice indices. In the following plots the values that represent the non-positive values obtained when computing the dice index are as previously described in page 23.



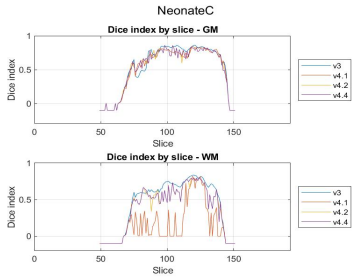
(a) Neonate7 - hierarchical



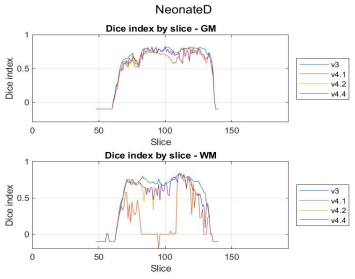
(b) NeonateA - hierarchical



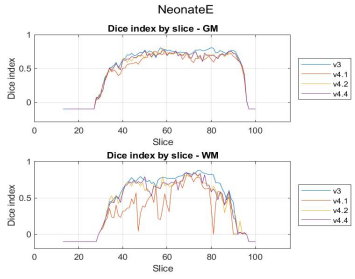
(c) NeonateB - hierarchical



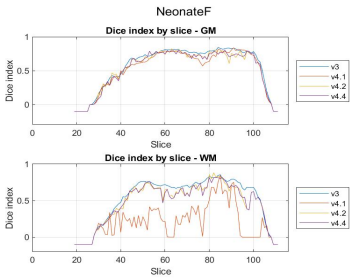
(d) NeonateC - hierarchical



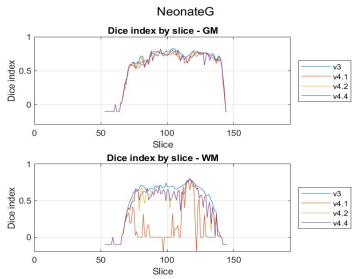
(e) NeonateD - hierarchical



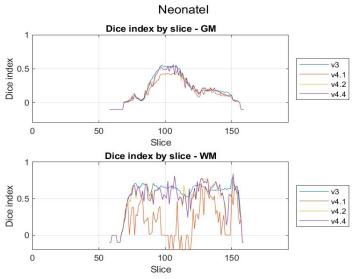
(f) NeonateE - hierarchical



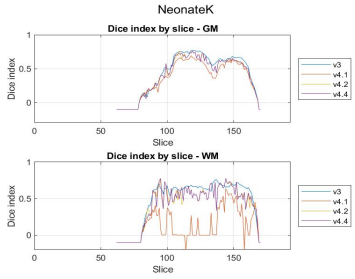
(g) NeonateF - hierarchical



(h) NeonateG - hierarchical

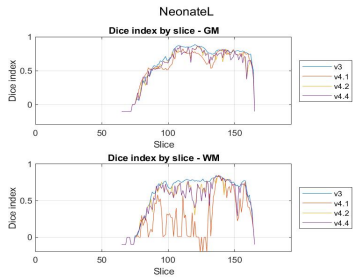


(i) NeonateI - hierarchical

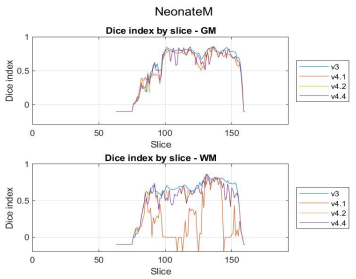


(j) NeonateK - hierarchical

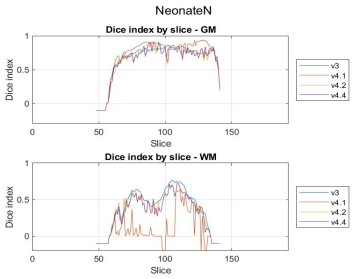
Figure 3.9: Hierarchical clustering results



(k) NeonateL - hierarchical



(l) NeonateM - hierarchical



(m) NeonateN - hierarchical

Figure 3.9: Hierarchical clustering results

Even if this version was done to hopefully improve the segmentation, in the obtained it can be seen how v3 has better results for both the GM and the WM mask in all patients.

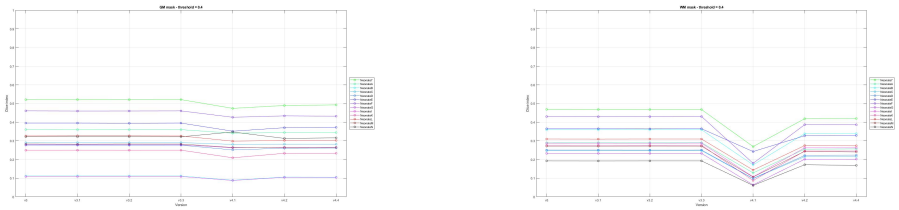
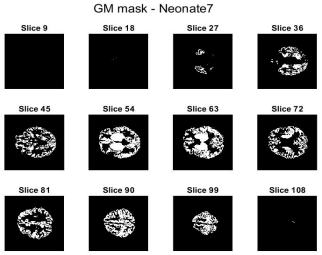


Figure 3.10: Code version comparison

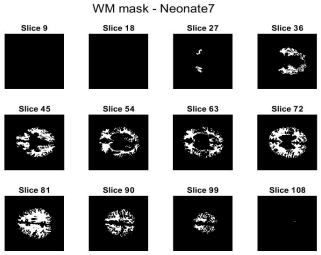
When comparing all code versions that had been obtained up until these last ones it can be seen that: for all versions both masks are better obtained with either v3 or v3.3 (really similar results). It also interesting to note that all the versions have lower dice indices for the WM mask (compared to the GM mask), this is seen clearly for version 4.1.

3.5 Patient selection

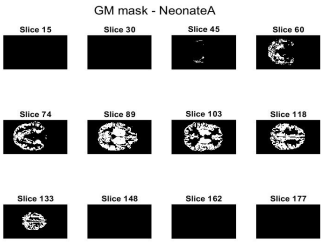
After obtaining unfavourable results and not being able to explain the inconsistencies obtained, a closer look at the standard segmentations was taken.



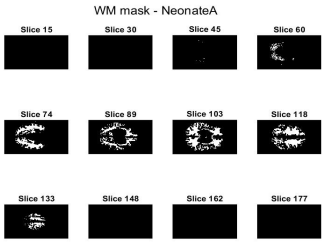
(a) Neonate7 - GM slices



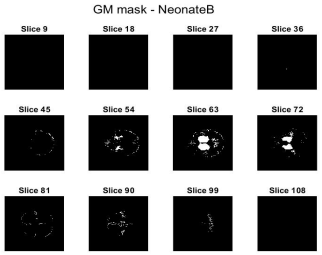
(b) Neonate7 - WM slices



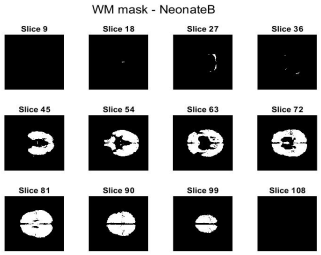
(c) NeonateA - GM slices



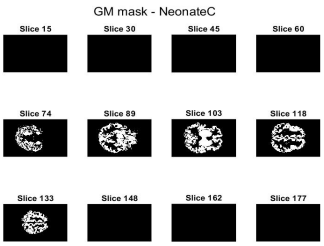
(d) NeonateA - WM slices



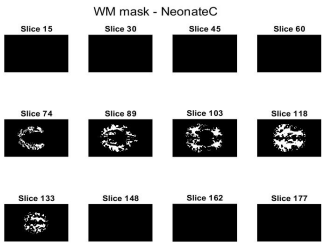
(e) NeonateB - GM slices



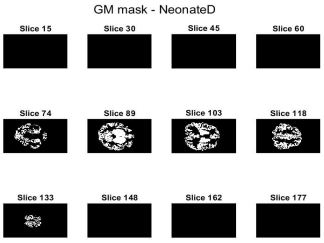
(f) NeonateB - WM slices



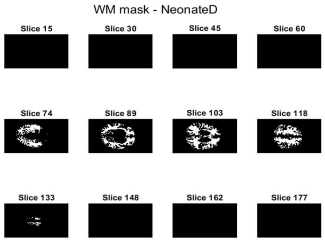
(g) NeonateC - GM slices



(h) NeonateC - WM slices

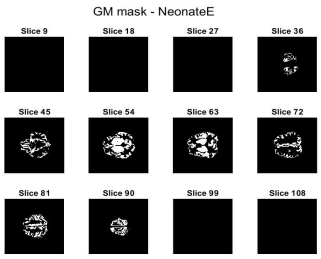


(i) NeonateD - GM slices

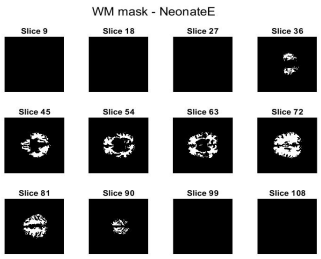


(j) NeonateD - WM slices

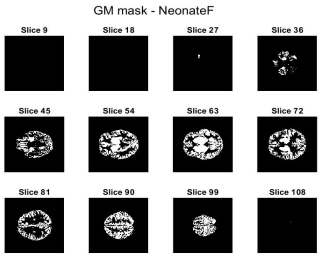
Figure 3.11: Patients selection: GM and WM display



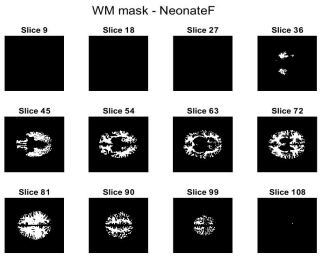
(k) NeonateE - GM slices



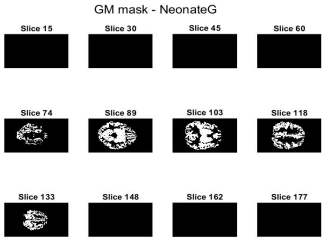
(l) NeonateE - WM slices



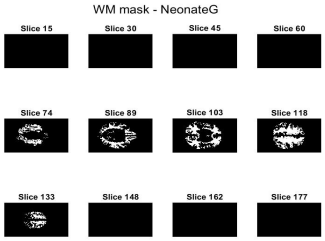
(m) NeonateF - GM slices



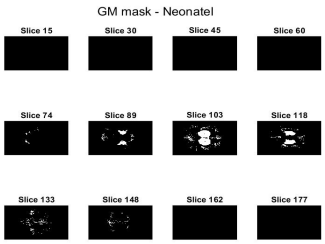
(n) NeonateF - WM slices



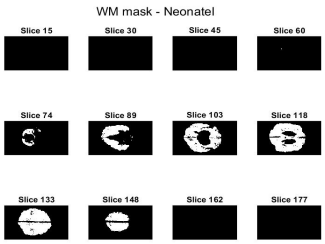
(o) NeonateG - GM slices



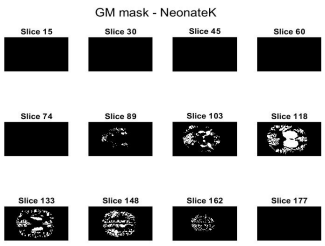
(p) NeonateG - WM slices



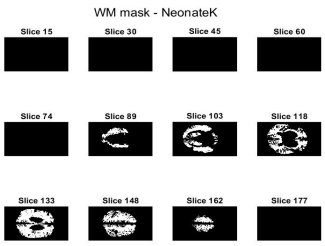
(q) NeonateI - GM slices



(r) NeonateI - WM slices

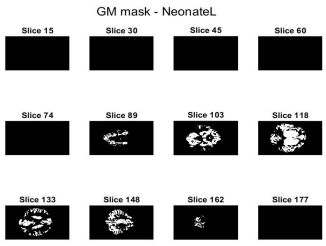


(s) NeonateK - GM slices

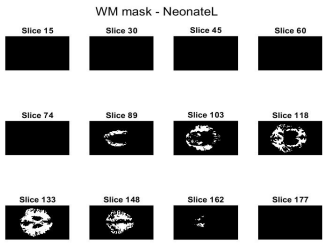


(t) NeonateK - WM slices

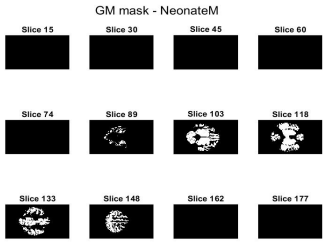
Figure 3.11: Patients selection: GM and WM display



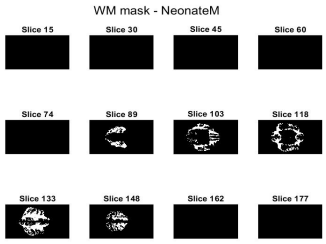
(u) NeonateL - GM slices



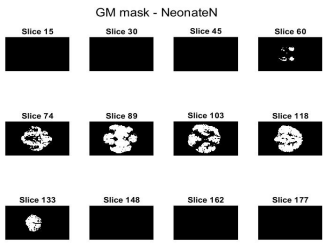
(v) NeonateL - WM slices



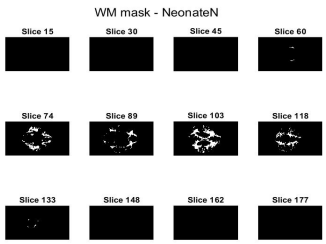
(w) NeonateM - GM slices



(x) NeonateM - WM slices



(y) NeonateN - GM slices



(z) NeonateN - WM slices

Figure 3.11: Patients selection: GM and WM display

As it can be seen in these images some standard images are not segmented accurately, this has a negative impact on the obtained results as they are no longer based on a true standard. To avoid making modification based on wrong results, all those masks that were considered not accurate were removed from the study group. So, after this point only seven subjects were considered. The discarded subjects were: Neonate B, Neonate I, Neonate K, Neonate L, Neonate M and Neonate N.

3.6 Dendrogram analysis

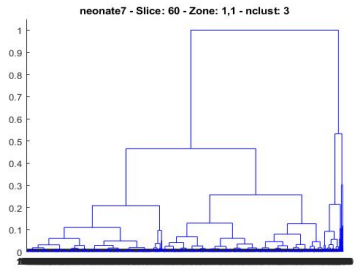
In the versions obtained with hierarchical clustering unfavourable results were obtained but further analysis was done to see if there could be a better way of using this type of clustering. This analysis was done by displaying the dendrograms of the central slice (the one with the most pixels) for each patient. This was done for versions 4.2 and 4.4 since v4.1 had really low indices.

The zones depend on the selected slice as follows:

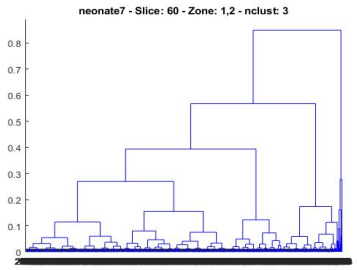
```
if nSlice_in<25 || nSlice_in>=95
    nZone=3;
elseif nSlice_in<=55
    nZone=9;
elseif nSlice_in<=75
    nZone=7;
else
    nZone=5;
end
```

Figure 3.12: Zone selection according to slice

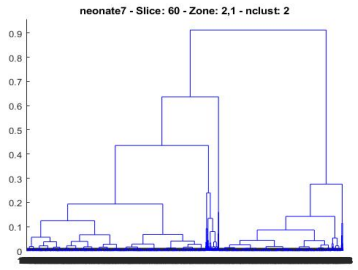
For each slice there are different zones depending on where it's situated inside the mask. The dendrograms were plotted for all zones. The following pictures show the dendrograms for Neonate7 (v4.2):



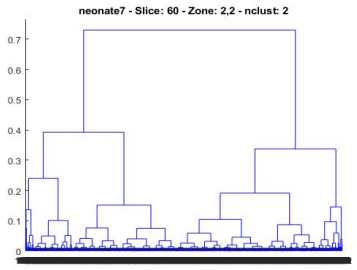
(a) Dendrogram - zone 1.1



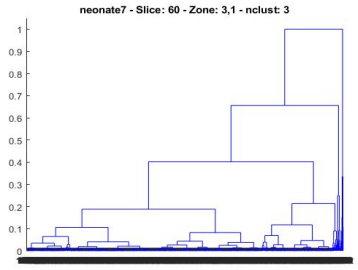
(b) Dendrogram - zone 1.2



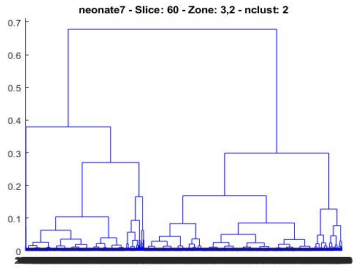
(c) Dendrogram - zone 2.1



(d) Dendrogram - zone 2.2

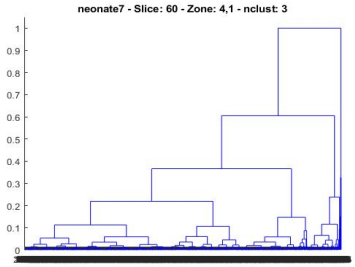


(e) Dendrogram - zone 3.1

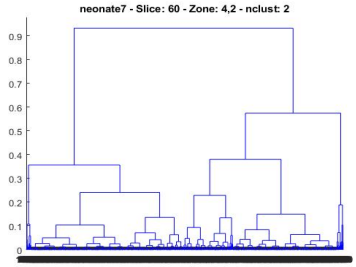


(f) Dendrogram - zone 3.2

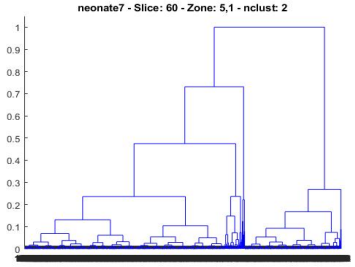
Figure 3.13: Dendrograms Neonate7



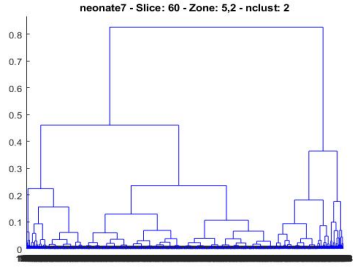
(g) Dendrogram - zone 4.1



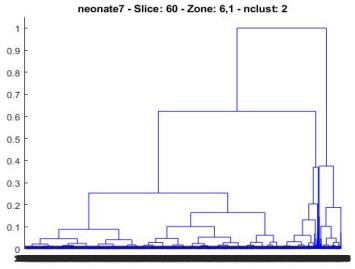
(h) Dendrogram - zone 4.2



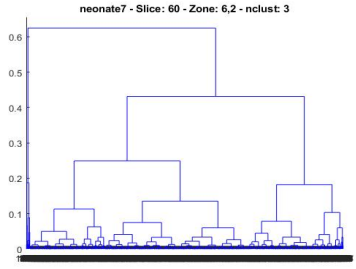
(i) Dendrogram - zone 5.1



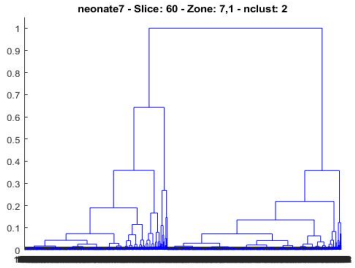
(j) Dendrogram - zone 5.2



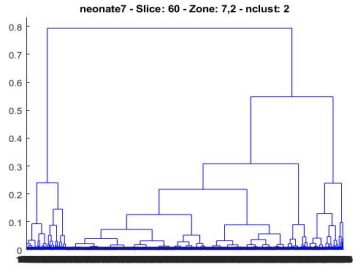
(k) Dendrogram - zone 6.1



(l) Dendrogram - zone 6.2



(m) Dendrogram - zone 7.1



(n) Dendrogram - zone 7.2

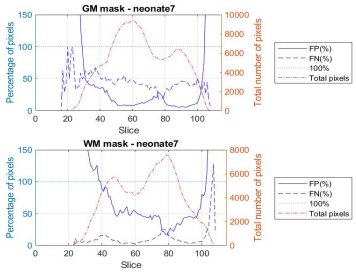
Figure 3.13: Dendrograms Neonate7

Looking at all the dendrograms, which are similar to the ones obtained for Neonate7, it can be seen that the maximum number of clusters in which the zone is divided is two. This number of clusters is clearly not enough to properly segment the image.

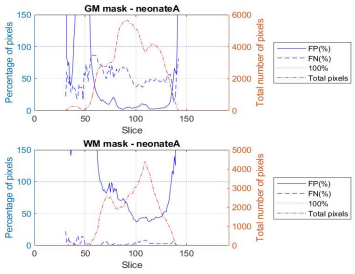
3.7 *clust_image* function improvement

One of the functions that is used to obtain the grey matter mask is the *Clust_image* function. In this function, the k-means or the hierarchical method are used to create the different clusters. After analysing v4.2 and v4.4 dendrograms, it was decided that having only one or two clusters for each section might not be enough. Consequently, said function was modified so that, according to each section's characteristics, more clusters could be obtained.

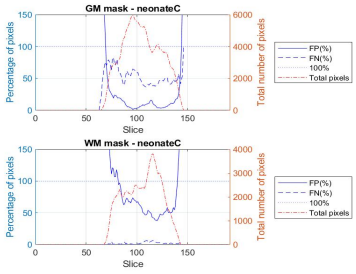
To check if this had a positive impact on the segmentation, three new code versions were created: v3.0.1, v4.2.1 and v4.4.1. In each version the *Clust_image* was changed with the new function.



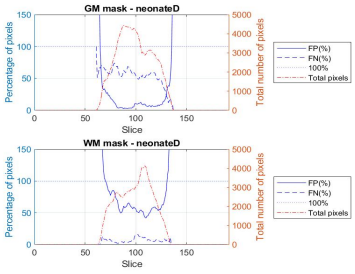
(a) Neonate7 - pixels percentage



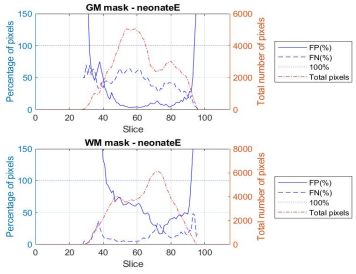
(b) NeonateA - pixels percentage



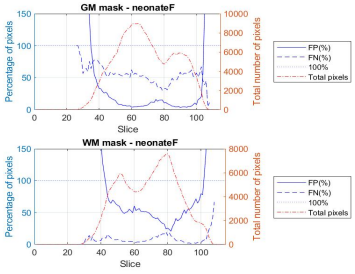
(c) NeonateC - pixels percentage



(d) NeonateD - pixels percentage

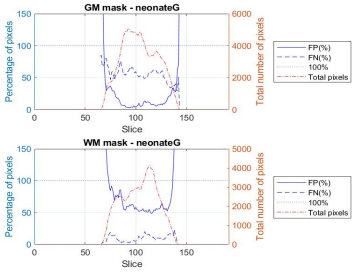


(e) NeonateE - pixels percentage



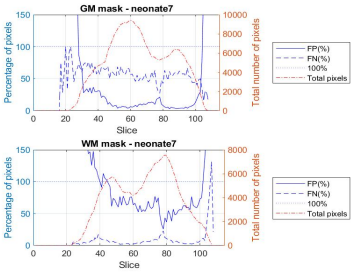
(f) NeonateF - pixels percentage

Figure 3.14: Pixel percentage - v3.0.1

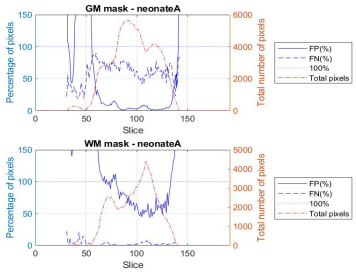


(g) NeonateG - pixels percentage

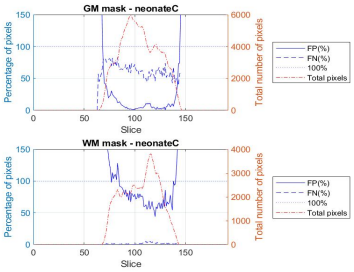
Figure 3.14: Pixel percentage - v3.0.1



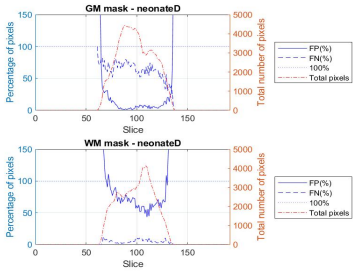
(a) Neonate7 - pixels percentage



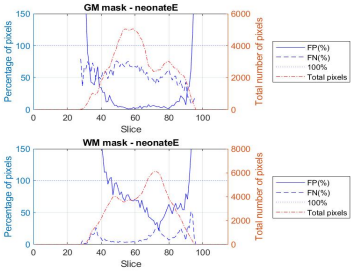
(b) NeonateA - pixels percentage



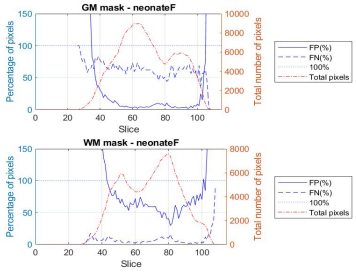
(c) NeonateC - pixels percentage



(d) NeonateD - pixels percentage

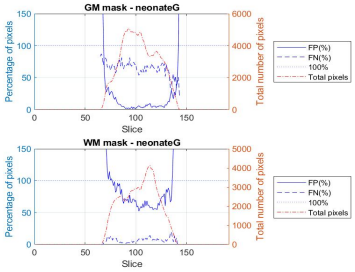


(e) NeonateE - pixels percentage



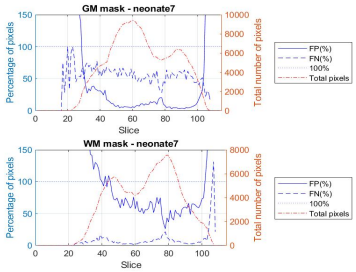
(f) NeonateF - pixels percentage

Figure 3.15: Pixel percentage - v4.2.1

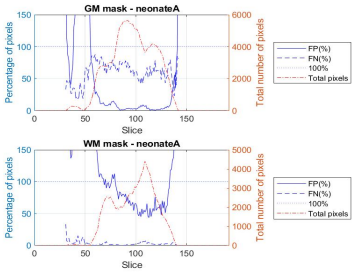


(g) NeonateG - pixels percentage

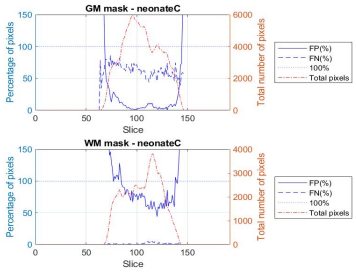
Figure 3.15: Pixel percentage - v4.2.1



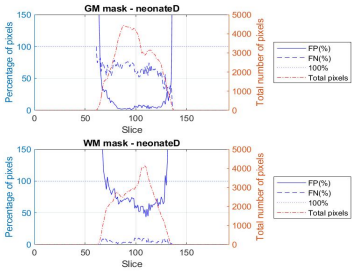
(a) Neonate7 - pixels percentage



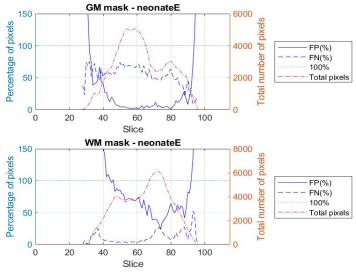
(b) NeonateA - pixels percentage



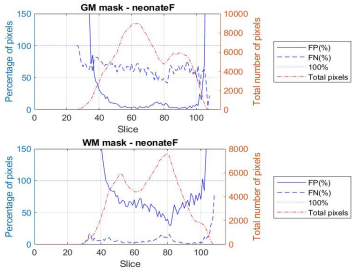
(c) NeonateC - pixels percentage



(d) NeonateD - pixels percentage

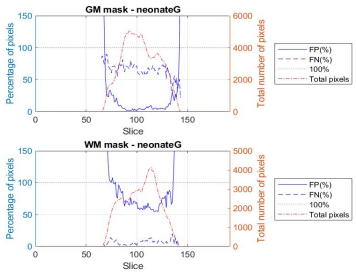


(e) NeonateE - pixels percentage



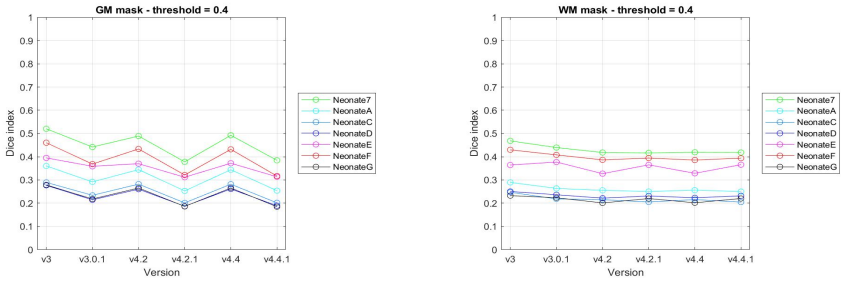
(f) NeonateF - pixels percentage

Figure 3.16: Pixel percentage - v4.4.1



(g) NeonateG - pixels percentage

Figure 3.16: Pixel percentage - v4.4.1



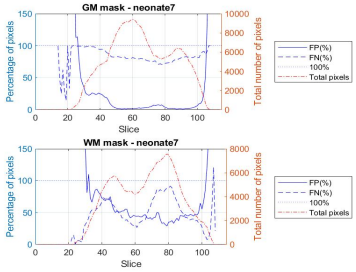
(a) Dice index - GM mask (b) Dice index - WM mask

Figure 3.17: Dice index comparison - function improvement

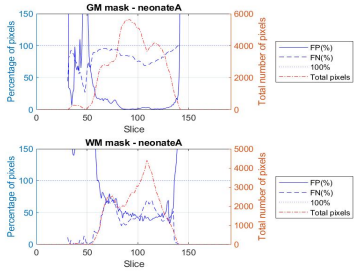
Unfortunately, after plotting the results, it can be seen that the new function actually entailed a decrease in the dice indices.

3.8 Segmentation order

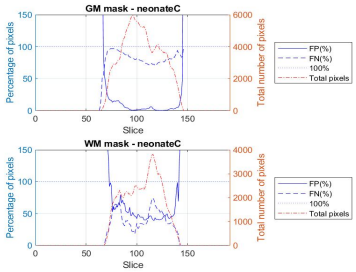
Seeing how the grey matter (GM) masks had higher indices than the white matter (WM) masks, a new code version (v5) was done to try to solve the issue. In previous versions, the GM mask was obtained before the WM mask, therefore in the new version the order was switched. To properly assign each pixel to its corresponding mask, v3 and v5 were compared pixel by pixel and then each pixel was assigned to the mask (GM or WM) which had the highest intensity, keeping said intensity.



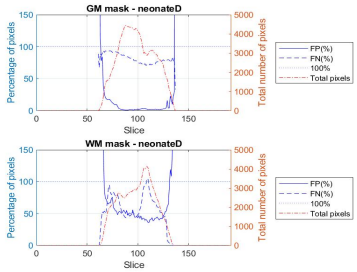
(a) Neonate7 - pixels percentage



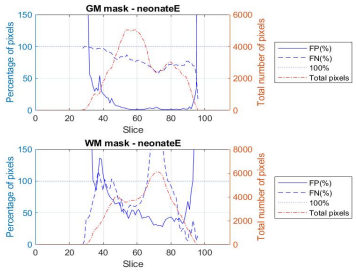
(b) NeonateA - pixels percentage



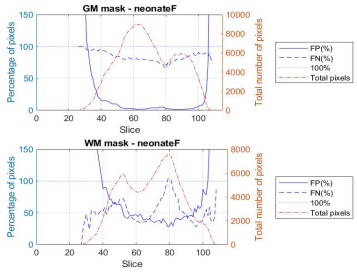
(c) NeonateC - pixels percentage



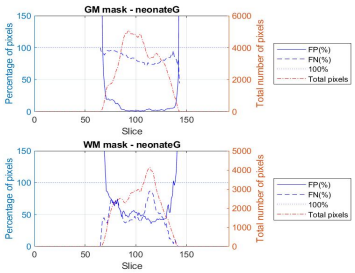
(d) NeonateD - pixels percentage



(e) NeonateE - pixels percentage



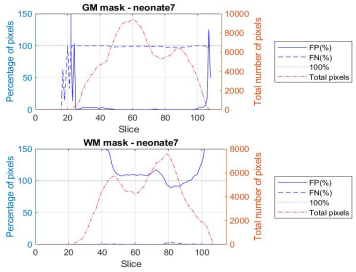
(f) NeonateF - pixels percentage



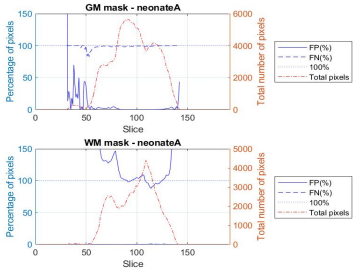
(g) NeonateG - pixels percentage

Figure 3.18: Pixel percentage - v5

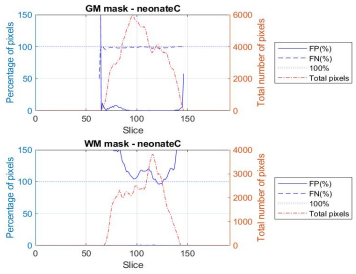
Another version (v5.1) was created where the order of obtaining the masks was switched from the original but the pixels were not compared to v3.



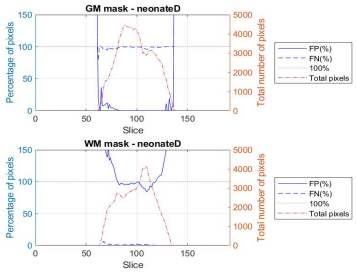
(a) Neonate7 - pixels percentage



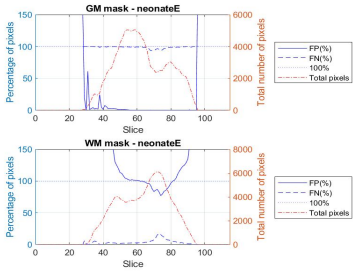
(b) NeonateA - pixels percentage



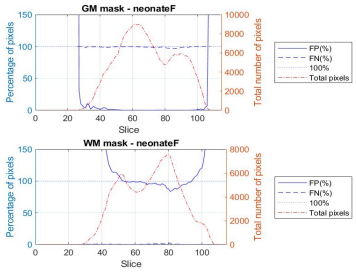
(c) NeonateC - pixels percentage



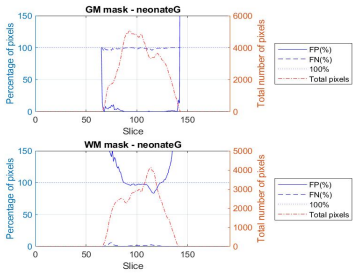
(d) NeonateD - pixels percentage



(e) NeonateE - pixels percentage



(f) NeonateF - pixels percentage



(g) NeonateG - pixels percentage

Figure 3.19: Pixel percentage - v5.1

After comparing the obtained masks with the standard masks, these versions were discarded from further analysis because of the extremely low dice indices.

3.9 Final results

In the following tables, a recap of all the version's dice indices for the seven selected neonates can be found:

Table 3.4: GM masks' dice index

	7	A	C	D	E	F	G
v3	0.520	0.360	0.289	0.276	0.395	0.460	0.278
v3.0.1	0.441	0.291	0.234	0.214	0.358	0.368	0.218
v3.1	0.520	0.359	0.290	0.276	0.394	0.459	0.278
v3.2	0.520	0.359	0.290	0.276	0.393	0.459	0.278
v3.3	0.520	0.359	0.290	0.276	0.395	0.460	0.278
v4.1	0.473	0.340	0.280	0.251	0.351	0.426	0.262
v4.2	0.489	0.344	0.282	0.260	0.370	0.433	0.265
v4.2.1	0.378	0.253	0.201	0.187	0.311	0.321	0.186
v4.4	0.492	0.344	0.281	0.261	0.372	0.431	0.265
v4.4.1	0.384	0.254	0.201	0.189	0.313	0.317	0.184
v5	0.182	0.138	0.113	0.111	0.186	0.174	0.101
v5.1	0.020	0.013	0.009	0.010	0.014	0.011	0.009

Table 3.5: WM masks' dice index

	7	A	C	D	E	F	G
v3	0.468	0.290	0.247	0.251	0.364	0.430	0.233
v3.0.1	0.439	0.264	0.219	0.237	0.377	0.408	0.224
v3.1	0.468	0.291	0.248	0.251	0.364	0.429	0.233
v3.2	0.468	0.290	0.248	0.250	0.364	0.429	0.233
v3.3	0.468	0.290	0.248	0.251	0.365	0.430	0.233
v4.1	0.269	0.128	0.098	0.098	0.242	0.180	0.064
v4.2	0.418	0.255	0.215	0.222	0.327	0.386	0.201
v4.2.1	0.416	0.250	0.205	0.231	0.365	0.394	0.220
v4.4	0.419	0.256	0.216	0.223	0.329	0.386	0.202
v4.4.1	0.418	0.250	0.205	0.232	0.366	0.394	0.220
v5	0.213	0.100	0.079	0.104	0.192	0.190	0.116
v5.1	0.364	0.215	0.171	0.207	0.336	0.352	0.202

Chapter 4

Conclusions and future work

After deeply analysing all the results obtained from the first versions to the final ones, generated based on the previously obtained results, a better segmentation has been obtained. Through trial and error, different methods were discarded such as hierarchical clustering and *clust_image* function modification.

In conclusion, the best version for segmenting the dice is v3.3 which obtains the grey matter masks like v0 and the white matter masks like v0.1 and has both of the mask obtaining thresholds equal to 0.4. Even though it is the best version, the grey matter masks still have higher dice indices than the white matter masks.

It should be noted that even though the thesis started with thirteen test subjects, this number was reduced to seven since the standard masks were not properly segmented. In the future, the obtained results could be checked with better standard segmentations to reaffirm the obtained conclusions.

The main issue found when obtaining the masks, which negatively affects the segmentation and that should be solved in the future, is the processing of the first and last slices for each patient. It is in these slices where most of the false classifications happen. These slices are mainly empty in the standard but get a lot of pixel detection in the clustering masks. Were this to be solves, better dice indices could be obtained.

Appendices

Appendix A

main_perArticolo function

```
%clear all; close all;clc;

D = dir('\\192.168.164.132\ricerca_ehealth\4.Genova_MRI brain\immagini
    per articolo gennaio 2019');
D(1:2)=[];

for i =1:length(D)

    nomefile=D(i).name;

    % caricamento T1
    if i == 1 || 5
        pathT1 = ['\\192.168.164.132\ricerca_ehealth\4.Genova_MRI brain
    \immagini per articolo gennaio 2019\',nomefile,'\T1_bet.nii'];
    else
        pathT1 = ['\\192.168.164.132\ricerca_ehealth\4.Genova_MRI brain
    \immagini per articolo gennaio 2019\',nomefile,'\T1_bet.bfc.nii'];
    end
    T1 = niftiread(pathT1);
    T1 = double(T1);

    %caricamento FLAIR
    pathFLAIR=['\\192.168.164.132\ricerca_ehealth\4.Genova_MRI brain\
    immagini per articolo gennaio 2019\',nomefile,'\FLAIR_BET.nii'];
    FLAIR = niftiread(pathFLAIR);

    %caricamento T2
    pathT2=['\\192.168.164.132\ricerca_ehealth\4.Genova_MRI brain\
    immagini per articolo gennaio 2019\',nomefile,'\T2_bet.nii'];
    T2 = niftiread(pathT2);

    fprintf('%s:\n',nomefile)
    fprintf('\tT1:\t\t %d %d %d\n',size(T1,1),size(T1,2),size(T1,3))
    fprintf('\tT2:\t\t %d %d %d\n',size(T2,1),size(T2,2),size(T2,3))
    fprintf('\tFLAIR:\t %d %d %d\n\n',size(FLAIR,1),size(FLAIR,2),size(
    FLAIR,3))
```

```
 if size(FLAIR,1)~=size(T1,1) || size(FLAIR,2)~=size(T1,2) || size(
FLAIR,3)~=size(T1,3)
     keyboard
     FLAIR=im_reorientation(FLAIR,0);
     if size(FLAIR,1)==size(T1,1) && size(FLAIR,2)==size(T1,2) &&
size(FLAIR,3)==size(T1,3)
         niftiwrite(FLAIR,['\\192.168.164.132\ricerca_ehealth\4.
Genova_MRI brain\immagini per articolo\',nomefile,'\FLAIR_BET.nii
'])
     end
 end

 if size(T2,1)~=size(T1,1) || size(T2,2)~=size(T1,2) || size(T2,3)
~=size(T1,3)
     keyboard
     T2=im_reorientation(T2,0);
     if size(T2,1)==size(T1,1) && size(T2,2)==size(T1,2) && size(T2
,3)==size(T1,3)
         niftiwrite(T2,['\\192.168.164.132\ricerca_ehealth\4.
Genova_MRI brain\immagini per articolo\',nomefile,'\T2_BET.nii'])
     end
 end

%% T1 and FLAIR DENOISING
h=fspecial('average',3);
for z=1:size(T1,3)
    T1(:,:,z)=imfilter(T1(:,:,z),h);
end
h=fspecial('average',3);
for z=1:size(FLAIR,3)
    FLAIR(:,:,z)=imfilter(FLAIR(:,:,z),h);
end

%% SKULL REMOVAL
for z=1:size(T1,3)  %ciclo sulle slic
    SliceT1=T1(:,:,z);
    SliceT2=T2(:,:,z);
    SliceFLAIR=FLAIR(:,:,z);

    ind=find(SliceT1==0);
    SliceT2(ind)=0;
    SliceFLAIR(ind)=0;
    T2(:,:,z)=SliceT2;
    FLAIR(:,:,z)=SliceFLAIR;
end

%INIZIALIZATION
mask=zeros(size(T1,1),size(T1,2),size(T1,3),3);

for sl=1:size(T1,3)  %ciclo sulle slice
    nSlice=sl;

    SliceT1=T1(:,:,nSlice);
    SliceT2=T2(:,:,nSlice);
```

```
        if sum(SliceT1(:))>0 && sum(SliceT2(:))>0

            %% estrazione maschere
            [maskCSF]=calcolo_maschera_CSF1(T1,FLAIR,nSlice,0); %
    marging su slice adiacenti
            [maskGM]=calcolo_maschera_GM(T1,T2,nSlice,0,maskCSF);
            [maskWM]=calcolo_maschera_WM(T1,FLAIR,T2,nSlice,0,maskCSF+
    maskGM);

            mask(:,:,sl,1)=maskGM;
            mask(:,:,sl,2)=maskWM;
            mask(:,:,sl,3)=maskCSF;

            close all
        end
    end

    niftiwrite(mask,['\\192.168.164.132\ricerca_ehealth\4.Genova_MRI
    brain\risultati per articolo gennaio 2019\',nomefile,'_mask_v0_dis.
    nii'])
    close all
end
```

Appendix B

im_reorientation function

```
function image=im_reorientation(image_in,denoising)
%funzione per riorientare le immagini volumetriche dal piano sagittale
    al
%piano assiale
%
%image_in: immagine 3D da reorietare
%denoising: flag per effettuare il denoisining con filtro medio
%image: immagine riorientata

image=zeros(size(image_in,3), size(image_in,1), size(image_in,2));
%filtro medio (denoising)
h=fspecial('average',3);

for z=1:size(image_in,3)
    image_in(:,:,z)=imrotate(image_in(:,:,z),180);
    if denoising==1
        image_in(:,:,z)=imfilter(image_in(:,:,z),h);
    end
    for y=1:size(image_in,2)
        for x=1:size(image_in,1)
            image(z,x,y)=image_in(x,y,z);
        end
    end
end
```

Appendix C

calcolo_maschera_CSF1 function

```
function [mask_finale]=calcolo_maschera_CSF1(varargin)
% (T1,FLAIR,nSlice_in,visual_flag)

%% inizializzazioni
T1=varargin{1};
FLAIR=varargin{2};
nSlice_in=varargin{3};
count=1;
mask_slice=zeros(size(T1,1),size(T1,2),3);
%% ciclo su slice adiacenti
for nSlice=nSlice_in-1:nSlice_in+1
    % Seleziono solo la slice di interesse
    SliceT1=T1(:,:,nSlice);
    SliceFLAIR=FLAIR(:,:,nSlice);
    %     SliceMask=maskBrain(:,:,nSlice);

    % Verifico che la slice sia non vuota
    if sum(SliceT1(:))>0

        %% Crop
        xx=find(sum(SliceT1,1)>0);
        yy=find(sum(SliceT1,2)>0);
        if nSlice==nSlice_in
            xx_Sl=xx;
            yy_Sl=yy;
        end
        SliceT1_c = imcrop(SliceT1, [xx(1) yy(1) xx(end)-xx(1) yy(end)-
    yy(1)]);
        %             figure, imshow(SliceT1_c,[]), title('T1 cropped')
        SliceFLAIR_c = imcrop(SliceFLAIR, [xx(1) yy(1) xx(end)-xx(1) yy
    (end)-yy(1)]);

        SliceT1_c(SliceT1_c==0)=NaN;
        SliceFLAIR_c(SliceFLAIR_c==0)=NaN;

        if nSlice==nSlice_in && varargin{4}==1
```

```
            figure, imshow(SliceT1_c,[]), title('T1 cropped')
            set(gcf, 'Position', get(0, 'Screensize'));
            figure, imshow(SliceFLAIR_c,[]), title('FLAIR cropped')
            set(gcf, 'Position', get(0, 'Screensize'));
        end

        %% Normalizzazione Immagini
        SliceT1_c=(SliceT1_c-min(SliceT1_c(:)))/(max(SliceT1_c(:))-min(
    SliceT1_c(:)));
        SliceFLAIR_c=(SliceFLAIR_c-min(SliceFLAIR_c(:)))/(max(
    SliceFLAIR_c(:))-min(SliceFLAIR_c(:)));
        if sum(SliceT1_c(:),'omitnan')>0 && sum(SliceFLAIR_c(:),'
    omitnan')>0
            Tp=SliceT1_c.*SliceFLAIR_c;
            if nSlice==nSlice_in && varargin{4}==1
                figure, imshow(Tp,[]), title('T1*FLAIR')
                set(gcf, 'Position', get(0, 'Screensize'));
            end

            %         %clustering
            %         SliceT1_c(SliceT1_c==0)=NaN;
            %         SliceFLAIR_c(SliceFLAIR_c==0)=NaN;


            %         %% Creazione della matrice da clusterizzare
            %         matr_clust=creazione_matr_clustering(SliceT1_c,
    SliceFLAIR_c, Tp);

            %% Estrazione del tessuto da ciascun tipo di immagine
            im_idx=[4:5];
            maskT_im=zeros(size(SliceT1_c,1),size(SliceT1_c,2),length(
    im_idx)); %matrice contenente le maschere ottenute da ciascun tipo
    di immagine
            for jj=1:length(im_idx)
                j=im_idx(jj);
                switch j
                    case 3
                        s1=prctile(SliceT1_c(:),5);
                        s2=prctile(SliceT1_c(:),10);
                        %                                         figure,
    imhist(SliceT1_c)
                        mask_im=region_growing1(SliceT1_c,s1,s2);
                    case 4
                        I=SliceFLAIR_c;
                        %                       I=imadjust(I,[0 1], [0
    1],2);
                        %                       I=histeq(I,256);
                        %                       figure,imshow(I,[])
                        s1=prctile(I(:),10); %10
                        s2=prctile(I(:),15); %15
                        mask_im=region_growing1(SliceFLAIR_c,s1,s2);
                    case 5
                        I=Tp;
                        s1=prctile(I(:),5); %5
                        s2=prctile(I(:),15); %15
```

```
                        %                                          figure,
    imhist(SliceT1_c.*(1-SliceT2_c))
                        mask_im=region_growing1(Tp,s1,s2);
                end
                maskT_im(:,:,jj)=mask_im;
                %                           sum(mask_im(:))
            end

            %% voting sulle maschere ottenute e creazione della
    maschera complessiva
            maskT=sum(maskT_im,3);
            %         mask_tot(maskT>=.3)=maskT(maskT>=.3); %inutile
            %         mask_tot=maskT;

            %% inserisco i risultati delle maschere ottenute in una
    matrice 3D
            mask_slice(yy(1):yy(end),xx(1):xx(end),count)=maskT;
            count=count+1;
        else
            count=count+1;
        end
    end
end
%% riduco il peso delle slice precedente e successiva
mask_slice(:,:,1)=mask_slice(:,:,1)*.75;  %75
mask_slice(:,:,3)=mask_slice(:,:,3)*.75;  %75

%% unione dei risultati ottenuti su slice adiacenti
mask_finale=sum(mask_slice,3);
if sum(mask_finale(:))>0
    mask_finale=(mask_finale-min(mask_finale(:)))/(max(mask_finale(:))-
    min(mask_finale(:)));
    mask_finale(mask_finale<=.4)=0; %anche 0.4
    se = strel('square',2);
    mask_finale = imclose(mask_finale,se);
    % figure,imshow(maskWM_finale,[])

    %elimino aree troppo piccole
    sogliaArea=4;
    [ROILab, numROI] = bwlabel(mask_finale, 4); %numero le roi
    A  = regionprops(ROILab, 'area');     %calcolo l'area di ogni roi
    for jj=1:numROI
        if A(jj).Area<=sogliaArea
            [r,c]=find(ROILab==jj);
            for ii=1:length(r)
                mask_finale(r(ii),c(ii))=0;  %elimino le roi con meno
    di 25 px
            end
        end
    end

    %riempio i buchi
    mask_temp=1-mask_finale;
    mask_temp(T1(:,:,nSlice_in)==0)=NaN;
    mask_temp(mask_temp<1)=0;
```

```
    % figure,imshow(mask_temp,[]);
    sogliaArea=4;
    [ROILab, numROI] = bwlabel(mask_temp, 4); %numero le roi
    A  = regionprops(ROILab, 'area');     %calcolo l'area di ogni roi
    for jj=1:numROI
        if A(jj).Area<=sogliaArea
            [r,c]=find(ROILab==jj);
            contorno=mask_finale(min(r)-1:max(r)+1,min(c)-1:max(c)+1);
            contorno(contorno==0)=NaN;
            fill=mean(contorno(:),'omitnan');
            for ii=1:length(r)
                mask_finale(r(ii),c(ii))=fill;  %riempio i buchi
            end
        end
    end

    mask_finale_c = imcrop(mask_finale, [xx_Sl(1) yy_Sl(1) xx_Sl(end)-
    xx_Sl(1) yy_Sl(end)-yy_Sl(1)]);

    figure,imshow(mask_finale_c,[])
    set(gcf, 'Position', get(0, 'Screensize'));
end
```

Appendix D

calcolo_maschera_GM function

```
function [mask_finale]=calcolo_maschera_GM(varargin)
% (T1,T2,nSlice_in,visual_flag)

%% inizializzazioni
T1=varargin{1};
T2=varargin{2};
nSlice_in=varargin{3};
count=1;
mask_slice=zeros(size(T1,1),size(T1,2),3);

% if nSlice_in<=55 && nSlice_in>=25
%     nZone=9;
% elseif nSlice_in<=75
%     nZone=7;
% elseif nSlice_in<=95
%     nZone=5;
% else
%     nZone=3;
% end

if nSlice_in<25 || nSlice_in>=95
    nZone=3;
elseif nSlice_in<=55
    nZone=9;
elseif nSlice_in<=75
    nZone=7;  %7
else
    nZone=5;
end

%% ciclo su slice adiacenti
for nSlice=nSlice_in-1:nSlice_in+1
    % Seleziono solo la slice di interesse
    SliceT1=T1(:,:,nSlice);
    SliceT2=T2(:,:,nSlice);
    %     SliceMask=maskBrain(:,:,nSlice);
```

```
% Verifico che la slice sia non vuota
if sum(SliceT1(:))>0 && sum(SliceT2(:))>0

    %% Crop
    xx=find(sum(SliceT1,1)>0);
    yy=find(sum(SliceT1,2)>0);

    %% elimino la maschera precedente
    mask_prec=varargin{5};
    SliceT1(mask_prec>0)=NaN;
    SliceT2(mask_prec>0)=NaN;

    if nSlice==nSlice_in
        xx_Sl=xx;
        yy_Sl=yy;
    end
    SliceT1_c = imcrop(SliceT1, [xx(1) yy(1) xx(end)-xx(1) yy(end)-
yy(1)]);
    %             figure, imshow(SliceT1_c,[]), title('T1 cropped')
    SliceT2_c = imcrop(SliceT2, [xx(1) yy(1) xx(end)-xx(1) yy(end)-
yy(1)]);

    SliceT1_c(SliceT1_c==0)=NaN;
    SliceT2_c(SliceT2_c==0)=NaN;

    if nSlice==nSlice_in && varargin{4}==1
        figure, imshow(SliceT1_c,[]), title('T1 cropped')
        set(gcf, 'Position', get(0, 'Screensize'));
        figure, imshow(SliceT2_c,[]), title('T2 cropped')
        set(gcf, 'Position', get(0, 'Screensize'));
    end

    %% Normalizzazione Immagini
    SliceT1_c=(SliceT1_c-min(SliceT1_c(:)))/(max(SliceT1_c(:))-min(
SliceT1_c(:)));
    SliceT2_c=(SliceT2_c-min(SliceT2_c(:)))/(max(SliceT2_c(:))-min(
SliceT2_c(:)));

    %             sum(SliceT1_c(:),'omitnan')
    %             sum(SliceT2_c(:),'omitnan')
    Tp=SliceT1_c.*(1-SliceT2_c);
    if nSlice==nSlice_in && varargin{4}==1
        figure, imshow(Tp,[]), title('T1*(1-T2)')
        set(gcf, 'Position', get(0, 'Screensize'));
        %             [Gmag2, ~] = imgradient(Tp,'Sobel');
        %             figure, imshow(Gmag2,[]), title('gradiente')
        %             set(gcf, 'Position', get(0, 'Screensize'));
        %             figure, imshow((1-exp(SliceT2_c)),[]), title
('1-expT2')
        %             set(gcf, 'Position', get(0, 'Screensize'));
        %             figure, imshow(SliceT1_c./(exp(SliceT2_c))
,[]), title('T1/expT2')
        %             set(gcf, 'Position', get(0, 'Screensize'));
    end
```

```
    %% inizializzazione delle maschere del tessuto
    mask_tot=zeros(size(SliceT1_c));

    %% divido le immagini in parti
    %         nZone=7;
    matr_clust_partizione=creazione_matr_clustering(SliceT2_c);
    matr_clust_partizione1=[];
    for i=1:size(matr_clust_partizione,2)
        matr_clust_partizione1(:,i)=matr_clust_partizione(:,i)/max(
matr_clust_partizione(:,i));
    end

    if size(matr_clust_partizione1,1)>=nZone

        c=kmeans(matr_clust_partizione1,nZone,'Distance', '
sqeuclidean', 'Replicates',10,'MaxIter',500);
        mask=creazione_maschere(SliceT2_c,c, matr_clust_partizione,
 0);
        SliceT1_or=SliceT1_c;
        SliceT2_or=SliceT2_c;

        %clustering per ciascuna zona
        for i=1:nZone
            SliceT1_c=SliceT1_or.*mask(:,:,i);
            SliceT1_c(SliceT1_c==0)=NaN;

            SliceT2_c=SliceT2_or.*mask(:,:,i);
            SliceT2_c(SliceT2_c==0)=NaN;

            %% Normalizzazione singola zona
            SliceT1_c=(SliceT1_c-min(SliceT1_c(:)))/(max(SliceT1_c
(:))-min(SliceT1_c(:)));
            SliceT2_c=(SliceT2_c-min(SliceT2_c(:)))/(max(SliceT2_c
(:))-min(SliceT2_c(:)));

            if sum(SliceT1_c(:),'omitnan')>0 && sum(SliceT2_c(:),'
omitnan')>0
                %% Creazione della matrice da clusterizzare
                matr_clust=creazione_matr_clustering(SliceT1_c,
SliceT2_c, SliceT1_c.*(1-SliceT2_c),SliceT1_c./(exp(SliceT2_c)));

                %% Estrazione del tessuto da ciascun tipo di
immagine
                classi=2;
                im_idx=4:5; %T2, T1*(1-T2)
                maskT_im=zeros(size(SliceT2_c,1),size(SliceT2_c,2),
length(im_idx)); %matrice contenente le maschere ottenute da
ciascun tipo di immagine
                for jj=1:length(im_idx)
                    j=im_idx(jj);

                    %clustering per WM
                    clust=clust_image(matr_clust(:,j),classi,
0.05,1);
```

```
                        mask_im=creazione_maschere(SliceT2_c,clust,
    matr_clust, 0.05);

                        % calcolo medie dei cluster
                        media=calcolo_medie(matr_clust, 5, clust, 0.05)
    ;

                        % individuazione pixel WM
                        ind=media(end,1);
                        maskT_im(:,:,jj)=mask_im(:,:,ind);
                    end

                    %% voting sulle maschere ottenute e creazione della
     maschera complessiva
                    maskT=sum(maskT_im,3);
                    maskT=(maskT-min(maskT(:)))/(max(maskT(:))-min(
    maskT(:)));
                    mask_tot(maskT>=0.1)=maskT(maskT>=0.1); %inutile
                end
            end
            %% inserisco i risultati delle maschere ottenute in una
    matrice 3D
            mask_slice(yy(1):yy(end),xx(1):xx(end),count)=mask_tot;
            count=count+1;
        else
            count=count+1;
        end
    end
end
%% riduco il peso delle slice precedente e successiva
mask_slice(:,:,1)=mask_slice(:,:,1)*.75;
mask_slice(:,:,3)=mask_slice(:,:,3)*.75;

%% unione dei risultati ottenuti su slice adiacenti
mask_finale=sum(mask_slice,3);
mask_finale=(mask_finale-min(mask_finale(:)))/(max(mask_finale(:))-min(
    mask_finale(:)));
mask_finale(mask_finale<0.5)=0;  %0.5 per 2 immagini/0.4 per 3 immagini
se = strel('square',2);
if sum(mask_finale(:))>0
    mask_finale = imclose(mask_finale,se);

    %elimino roi piccole
    sogliaArea=4;
    [ROILab, numROI] = bwlabel(mask_finale, 4); %numero le roi
    A  = regionprops(ROILab, 'area');     %calcolo l'area di ogni roi
    for jj=1:numROI
        if A(jj).Area<=sogliaArea
            [r,c]=find(ROILab==jj);
            for ii=1:length(r)
                mask_finale(r(ii),c(ii))=0;  %elimino le roi con meno
    di 25 px
            end
        end
    end
```

```
    %riempio i buchi
    mask_temp=1-mask_finale;
    mask_temp(T2(:,:,nSlice_in)==0)=NaN;
    mask_temp(mask_temp<1)=0;
    % figure,imshow(mask_temp,[]);
    sogliaArea=4;
    [ROILab, numROI] = bwlabel(mask_temp, 4); %numero le roi
    A  = regionprops(ROILab, 'area');     %calcolo l'area di ogni roi
    for jj=1:numROI
        if A(jj).Area<=sogliaArea
            [r,c]=find(ROILab==jj);
            contorno=mask_finale(min(r)-1:max(r)+1,min(c)-1:max(c)+1);
            contorno(contorno==0)=NaN;
            fill=mean(contorno(:),'omitnan');
            for ii=1:length(r)
                mask_finale(r(ii),c(ii))=fill;  %riempio i buchi
            end
        end
    end

    mask_finale_c = imcrop(mask_finale, [xx_Sl(1) yy_Sl(1) xx_Sl(end)-
    xx_Sl(1) yy_Sl(end)-yy_Sl(1)]);

    figure,imshow(mask_finale_c,[])
    set(gcf, 'Position', get(0, 'Screensize'));
end
```

Appendix E

calcolo_maschera_WM function

```
function [mask_finale]=calcolo_maschera_WM(varargin)
% (T1,FLAIR,nSlice_in,visual_flag)

%% inizializzazioni
T1=varargin{1};
FLAIR=varargin{2};
T2=varargin{3};
nSlice_in=varargin{4};
count=1;
mask_slice=zeros(size(T1,1),size(T1,2),3);
%% ciclo su slice adiacenti
for nSlice=nSlice_in-1:nSlice_in+1
    % Seleziono solo la slice di interesse
    SliceT1=T1(:,:,nSlice);
    SliceT2=T2(:,:,nSlice);
    SliceFLAIR=FLAIR(:,:,nSlice);
    %     SliceMask=maskBrain(:,:,nSlice);

    % Verifico che la slice sia non vuota
    if sum(SliceT1(:))>0  && sum(SliceT2(:))>0

        %% Crop
        xx=find(sum(SliceT1,1)>0);
        yy=find(sum(SliceT1,2)>0);

        %% elimino la maschera precedente
        mask_prec=varargin{6};
        SliceT1(mask_prec>0)=NaN;
        SliceT2(mask_prec>0)=NaN;
        SliceFLAIR(mask_prec>0)=NaN;

        if nSlice==nSlice_in
            xx_Sl=xx;
            yy_Sl=yy;
        end
```

```
    SliceT1_c = imcrop(SliceT1, [xx(1) yy(1) xx(end)-xx(1) yy(end)-
yy(1)]);
    %             figure, imshow(SliceT1_c,[]), title('T1 cropped')
    SliceFLAIR_c = imcrop(SliceFLAIR, [xx(1) yy(1) xx(end)-xx(1) yy
(end)-yy(1)]);
    SliceT2_c = imcrop(SliceT2, [xx(1) yy(1) xx(end)-xx(1) yy(end)-
yy(1)]);

    SliceT1_c(SliceT1_c==0)=NaN;
    SliceFLAIR_c(SliceFLAIR_c==0)=NaN;
    SliceT2_c(SliceT2_c==0)=NaN;

    if nSlice==nSlice_in && varargin{5}==1
        figure, imshow(SliceT1_c,[]), title('T1 cropped')
        set(gcf, 'Position', get(0, 'Screensize'));
        figure, imshow(SliceFLAIR_c,[]), title('FLAIR cropped')
        set(gcf, 'Position', get(0, 'Screensize'));
    end

    %% Normalizzazione Immagini
    SliceT1_c=(SliceT1_c-min(SliceT1_c(:)))/(max(SliceT1_c(:))-min(
SliceT1_c(:)));
    SliceT2_c=(SliceT2_c-min(SliceT2_c(:)))/(max(SliceT2_c(:))-min(
SliceT2_c(:)));
    SliceFLAIR_c=(SliceFLAIR_c-min(SliceFLAIR_c(:)))/(max(
SliceFLAIR_c(:))-min(SliceFLAIR_c(:)));

    Tp=SliceT1_c.*SliceFLAIR_c;
    if nSlice==nSlice_in && varargin{5}==1
        figure, imshow(Tp,[]), title('T1*FLAIR')
        set(gcf, 'Position', get(0, 'Screensize'));
    end

    %% inizializzazione della maschera del tessuto
    mask_tot=zeros(size(SliceT1_c));

    %         %clustering
    %         SliceT1_c(SliceT1_c==0)=NaN;
    %         SliceT2_c(SliceT2_c==0)=NaN;
    %         SliceFLAIR_c(SliceFLAIR_c==0)=NaN;


    %         %% Creazione della matrice da clusterizzare
    %         matr_clust=creazione_matr_clustering(SliceT1_c,
SliceFLAIR_c, Tp);

    %% Estrazione del tessuto da ciascun tipo di immagine
    im_idx=[3,5];
    maskT_im=zeros(size(SliceT1_c,1),size(SliceT1_c,2),length(
im_idx)); %matrice contenente le maschere ottenute da ciascun tipo
di immagine
    for jj=1:length(im_idx)
        j=im_idx(jj);
        switch j
            case 3 %T2
```

```
                    s1=prctile(SliceT2_c(:),50);
                    s2=prctile(SliceT2_c(:),5);
                    %                     s2=0;
                    %                                         figure,
    imhist(SliceT1_c)
                    mask_im=region_growing(SliceT2_c,s1,s2);
                case 4 %T1*FLAIR
                    s1=prctile(Tp(:),20);
                    s2=prctile(Tp(:),90);
                    %                     s2=0;
                    %                                         figure,
    imhist(SliceT1_c./(exp(SliceT2_c)))
                    mask_im=region_growing(Tp,s1,s2);
                case 5 %T1*(1-T2)
                    I=SliceT1_c.*(1-SliceT2_c);
                    s1=prctile(I(:),20);
                    s2=prctile(I(:),90);
                    %                     s2=0;
                    %                                         figure,
    imhist(SliceT1_c.*(1-SliceT2_c))
                    mask_im=region_growing1(I,s1,s2);
            end
            maskT_im(:,:,jj)=mask_im;
            %                           sum(mask_im(:))
        end

        %% voting sulle maschere ottenute e creazione della maschera
    complessiva
        maskT=sum(maskT_im,3);
        mask_tot(maskT>=.3)=maskT(maskT>=.3); %inutile

        %% inserisco i risultati delle maschere ottenute in una matrice
     3D
        mask_slice(yy(1):yy(end),xx(1):xx(end),count)=mask_tot;
        count=count+1;
    else
        count=count+1;
    end
end
%% riduco il peso delle slice precedente e successiva
mask_slice(:,:,1)=mask_slice(:,:,1)*.75;
mask_slice(:,:,3)=mask_slice(:,:,3)*.75;

%% unione dei risultati ottenuti su slice adiacenti
mask_finale=sum(mask_slice,3);
mask_finale=(mask_finale-min(mask_finale(:)))/(max(mask_finale(:))-min(
    mask_finale(:)));
mask_finale(mask_finale<=.4)=0;
if sum(mask_finale(:))>0
    se = strel('square',2);
    mask_finale = imclose(mask_finale,se);
    % figure,imshow(maskWM_finale,[])

    %elimino aree troppo piccole
    sogliaArea=4;
```

```
    [ROILab, numROI] = bwlabel(mask_finale, 4); %numero le roi
    A  = regionprops(ROILab, 'area');     %calcolo l'area di ogni roi
    for jj=1:numROI
        if A(jj).Area<=sogliaArea
            [r,c]=find(ROILab==jj);
            for ii=1:length(r)
                mask_finale(r(ii),c(ii))=0;  %elimino le roi con meno
    di 25 px
            end
        end
    end

    %riempio i buchi
    mask_temp=1-mask_finale;
    mask_temp(T1(:,:,nSlice_in)==0)=NaN;
    mask_temp(mask_temp<1)=0;
    % figure,imshow(mask_temp,[]);
    sogliaArea=4;
    [ROILab, numROI] = bwlabel(mask_temp, 4); %numero le roi
    A  = regionprops(ROILab, 'area');     %calcolo l'area di ogni roi
    for jj=1:numROI
        if A(jj).Area<=sogliaArea
            [r,c]=find(ROILab==jj);
            contorno=mask_finale(min(r)-1:max(r)+1,min(c)-1:max(c)+1);
            contorno(contorno==0)=NaN;
            fill=mean(contorno(:),'omitnan');
            for ii=1:length(r)
                mask_finale(r(ii),c(ii))=fill;  %riempio i buchi
            end
        end
    end

    mask_finale_c = imcrop(mask_finale, [xx_Sl(1) yy_Sl(1) xx_Sl(end)-
    xx_Sl(1) yy_Sl(end)-yy_Sl(1)]);

    figure,imshow(mask_finale_c,[])
    set(gcf, 'Position', get(0, 'Screensize'));
end
% for i =1:9
%     matr_clust_t(i).Sl1=scarta_pixel(matr_clust_t(i).Sl1,
    mask_finale_c,0);
%     matr_clust_t(i).Sl2=scarta_pixel(matr_clust_t(i).Sl2,
    mask_finale_c,0);
%     matr_clust_t(i).Sl3=scarta_pixel(matr_clust_t(i).Sl3,
    mask_finale_c,0);
% end
```

Appendix F

creazione_matr_clustering function

```
function matr_clust=creazione_matr_clustering(varargin)
%funzione per creare la matrice da usare per il clustering a partire da
     una
%serie di immagini
%
%input: elenco di immagini che si vogliono usare per costruire la
    matrice
%       per il clustering
%matr_clust: matrice con nrighe=numero di pixel validi (no NaN) nelle
%            immagini analizzate e ncolonne=coordX, coordY, nro di
    immagini analizzate

ind=1; %contatore
n=nargin; %numero di immagini date in input
matr_clust=zeros(1,n+2);

I1=varargin{1};
for i=1:size(I1,1)
    for j=1:size(I1,2)
        if ~isnan(I1(i,j))%se il pixel analizzato non e NaN lo
    inserisco nella matrice
            matr_clust(ind,1:2)=[i j]; %coordX, coordY
            for k=1:n %analizzo tutte le immagini
                I=varargin{k};
                matr_clust(ind,k+2)=I(i,j);
            end
            ind=ind+1;
        end
    end
end
```

Appendix G

region_growing1 function

```
function K2=region_growing1(I,T1,T2)

%Find seed points, i.e. pixels above threshold T1
seeds = find(I<=T1);

%Show image with first threshold applied, by putting the seed values to
     1
K =  zeros(size(I));
K(seeds)=1;
% figure
% imshow(K)
% title('image with first threshold applied')

%Create a vector ToProcess which will contain the linear indexes of the
%pixels that need to be processed one at a time.
ToProcess =  seeds; %Queue

%create segmentation mask, initially a matrix of zeros
K2 = zeros(size(K));

%While there are still elements that need to be processed, do the
    following
%loop:
while ~isempty(ToProcess) %Insert while condition

    %Create variable "current" which will contain the linear index of
    the
    %current pixel that is being processed
    current =  ToProcess(1);

    %Update K2
    %Put the current pixel in the mask K2 to 1:
    K2(current) = 1;

    %retrieve row and column of current pixel (see sub2ind)
    [r,c] = ind2sub(size(K2),current);
```

```
    %Get 8-neighbours of the current pixel
    %should check for borders of image;
    for i=r-1:r+1
        for j=c-1:c+1
            if i>0 && i<=size(K2,1) && j>0 && j<=size(K2,2)

    %Check to see if neighbors are above threshold T2, and remember not
     to
    %reprocess already processed pixels!!!
                if I(i,j)<=T2 && K(i,j)==0
                     %Update vector ToProcess, which should eliminate
    the pixel that was
    %just processed and add the neighbors of the processed pixel that
    %satisfy the two previous conditions.
    %Remember to reconvert indexes from subscript to linear!
                    ToProcess(end+1)=sub2ind(size(K2),i,j);
                    K2(i,j)=1;
                end
                K(i,j)=1; %pixel processato
            end
        end
    end
    ToProcess(1)=[];
end
```

Appendix H

region_growing function

```
function K2=region_growing(I,T1,T2)

%Find seed points, i.e. pixels above threshold T1
seeds = find(I>=T1);

%Show image with first threshold applied, by putting the seed values to
     1
K =  zeros(size(I));
K(seeds)=1;
% figure
% imshow(K)
% title('image with first threshold applied')

%Create a vector ToProcess which will contain the linear indexes of the
%pixels that need to be processed one at a time.
ToProcess =  seeds; %Queue

%create segmentation mask, initially a matrix of zeros
K2 = zeros(size(K));

%While there are still elements that need to be processed, do the
    following
%loop:
while ~isempty(ToProcess) %Insert while condition

    %Create variable "current" which will contain the linear index of
    the
    %current pixel that is being processed
    current =  ToProcess(1);

    %Update K2
    %Put the current pixel in the mask K2 to 1:
    K2(current) = 1;

    %retrieve row and column of current pixel (see sub2ind)
    [r,c] = ind2sub(size(K2),current);
```

```
    %Get 8-neighbours of the current pixel
    %should check for borders of image;
    for i=r-1:r+1
        for j=c-1:c+1
            if i>0 && i<=size(K2,1) && j>0 && j<=size(K2,2)

    %Check to see if neighbors are above threshold T2, and remember not
     to
    %reprocess already processed pixels!!!
                if I(i,j)>=T2 && K(i,j)==0
                     %Update vector ToProcess, which should eliminate
    the pixel that was
    %just processed and add the neighbors of the processed pixel that
    %satisfy the two previous conditions.
    %Remember to reconvert indexes from subscript to linear!
                    ToProcess(end+1)=sub2ind(size(K2),i,j);
                    K2(i,j)=1;
                end
                K(i,j)=1; %pixel processato
            end
        end
    end
    ToProcess(1)=[];
end
```

Appendix I

scarta_pixel function

```
function matr_clust=scarta_pixel(matr_clust,mask,soglia)
%funzione per scartare dalla matrice da usare per il clustering i pixel
     gia
%inclusi nel tessuto precedente
%
%matr_clust: matrice usata per il clustering
%mask: maschera del tessuto di cui si vogliono scartare i pixel
%soglia: soglia su mask per individuare i pixel apparteneti al tessuto
    da
%scartare

[x,y]=find(mask>soglia);
for i=1:length(x)
   ind=find(matr_clust(:,1)==x(i) & matr_clust(:,2)==y(i));
   matr_clust(ind,:)=[];
end
```

Appendix J

calcolo_medie function

```
function media=calcolo_medie(matr_clust, col, clust,soglia)
%funzione per il calcolo dei valori medi dei pixel nei cluster ottenuti
%
%matr_clust: matrice usata per il clustering
%col: colonna di matr_clust rispetto a cui calcolare la media
%clust: risultato clusterizzazione
%media: matr con nrighe=ncluster e 2 colonne (indice cluster, valor
    medio)
%       e ordinata in ordine crescente di valor medio

ncluster=max(clust); %nro di cluster
media=zeros(ncluster,2);
ind_rej=[]; %indici dei cluster da eliminare perche troppo piccoli
for i =1:ncluster
    nElem=length(find(clust==i));
    if nElem<length(clust)*soglia
        ind_rej=[ind_rej;i];
    end
    media(i,1)=i;
    media(i,2)=mean(matr_clust(clust==i,col));
end
media(ind_rej,:)=[];
media=sortrows(media,2);
```

Appendix K

creazione_maschere function

```
function mask=creazione_maschere(image,clust, matr_clust, visual_mask)
%funzione per la creazione di maschere separate per ciascun cluster
%individuato e eventuale plot delle maschere
%
%image= immagine di riferimento
%clust= risultato clustering
%matr_clust= matrice usata per il clustering
%visual_mask= flag per la visualizzazione delle maschere sull'immagine
%mask= matrice 3D in cui la terza dimensione rappresenta la maschera
%estratta da ciascun cluster (si considerano solo i cluster con almeno
    il
%5% dei pixel totali

ncluster=max(clust); %nro di cluster
mask=zeros([size(image),ncluster]); %matrice che conterra le maschere
col=hsv(ncluster);

for k=1:ncluster
    cl=find(clust==k);
%     if length(cl)>length(clust)*0.05
        if visual_mask==1
            figure
            imshow(image,[]);
            hold on
            plot(matr_clust(cl,2),matr_clust(cl,1),'*','color',col(k,:)
    )
        end
        masktmp=mask(:,:,k);
        masktmp(sub2ind(size(image),matr_clust(cl,1),matr_clust(cl,2)))
    =1;
        mask(:,:,k)=masktmp;
%     end
end
```

Appendix L

clust_image function

```
function clust=clust_image(matr_clust,nclust,soglia, centroids_flag)
%funzione per la clusterizzazione tramite kmeans dei valori in
    matr_clust
%in un numero di cluster pari a nclust. Se la clusterizzazione
    restutuisce
%dei cluster troppo piccoli (nElem<5% Elem tot), nclust viene
    incrementato
%di 1.
%centroids_flag=1 -> uso i percentili per inizializzare i centroidi

if ~centroids_flag
    clust=kmeans(matr_clust,nclust,'Replicates',100);
else
    percentili=[0:100/(nclust-1):100];
    centroids=prctile(matr_clust,percentili);
    clust=kmeans(matr_clust,nclust,'Start',centroids');
end

nElem=zeros(1,nclust); %nro di elementi per ogni cluster
for i=1:nclust
    nElem(i)=length(find(clust==i));
end

%verifico che non ci siano cluster troppo piccoli
if sum(nElem<length(clust)*soglia)>0
    nclust=nclust+1;
    if ~centroids_flag
        clust=kmeans(matr_clust,nclust,'Replicates',100);
    else
        percentili=[0:100/(nclust-1):100];
        centroids=prctile(matr_clust,percentili);
        clust=kmeans(matr_clust,nclust,'Start',centroids');
    end
end
```

Bibliography

[1] Aida Syafiqah Ahmad Khaizi, Rasyiqah Annani Mohd Rosidi, Hong-Seng Gan, and Khairil Amir Sayuti. A mini review on the design of interactive tool for medical image segmentation. In *2017 International Conference on Engineering Technology and Technopreneurship (ICE2T)*, pages 1–5. IEEE, 2017.

[2] Annemie Ribbens, Jeroen Hermans, Frederik Maes, Dirk Vandermeulen, and Paul Suetens. Unsupervised segmentation, clustering, and groupwise registration of heterogeneous populations of brain mr images. *IEEE transactions on medical imaging*, 33(2):201–224, 2013.

[3] Li Wang, Feng Shi, Gang Li, Yaozong Gao, Weili Lin, John H Gilmore, and Dinggang Shen. Segmentation of neonatal brain mr images using patch-driven level sets. *NeuroImage*, 84:141–158, 2014.

[4] Shruti Kapil, Meenu Chawla, and Mohd Dilshad Ansari. On k-means data clustering algorithm with genetic algorithm. In *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 202–206. IEEE, 2016.

[5] Shilpa Kamdi and RK Krishna. Image segmentation and region growing algorithm. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, 2(1), 2012.

[6] Puneet Jai Kaur et al. Cluster quality based performance evaluation of hierarchical clustering method. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, pages 649–653. IEEE, 2015.

[7] Alonso Gragera and Vorapong Suppakitpaisarn. Semimetric properties of sørensen-dice and tversky indexes. In *International Workshop on Algorithms and Computation*, pages 339–350. Springer, 2016.